Analytics on Indoor Moving Objects with Applications in Airport Baggage Tracking

Ahmed, Tanvir

DOI (link to publication from Publisher):
10.5278/vbn.phd.engsci.00073

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

ANALYTICS ON INDOOR MOVING OBJECTS WITH APPLICATIONS IN AIRPORT BAGGAGE TRACKING

BY
TANVIR AHMED

DISSERTATION SUBMITTED 2016

AALBORG UNIVERSITY
DENMARK

Analytics on Indoor Moving Objects with Applications in Airport Baggage Tracking

Ph.D. Dissertation

Tanvir Ahmed

Dissertation submitted March, 2016

A thesis submitted to the Faculty of Engineering and Science at Aalborg University (AAU) and the Faculty of Engineering at Université Libre De Bruxelles (ULB), in partial fulfillment of the requirements within the scope of the IT4BI-DC programme for the joint Ph.D. degree in computer science. The thesis is not submitted to any other organization at the same time.

Dissertation submitted: March, 2016

PhD supervisor: Prof. Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Assistant PhD supervisor: Assoc. Prof. Toon Calders
Université Libre de Bruxelles, Brussels, Belgium

PhD committee: Associate Professor Kristian Torp (chairman)
Aalborg University, Denmark

Associate Professor Peer Christian Kröger
Ludwig-Maximilians-Universität München, Germany

Associate Professor Alberto Abello
Universitat Politécnica de Catalunya, Spain

PhD Series: Faculty of Engineering and Science, Aalborg University

ISSN (online): 2246-1248
ISBN (online): 978-87-7112-529-0

Published by:
Aalborg University Press
Skjernvej 4A, 2nd floor
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2016

Abstract

A large part of people's lives are spent in indoor spaces such as office and university buildings, shopping malls, subway stations, airports, museums, community centers, etc. Such kind of spaces can be very large and paths inside the locations can be constrained and complex. Deployment of indoor tracking technologies like RFID, Bluetooth, and Wi-Fi can track people and object movements from one symbolic location to another within the indoor spaces. The resulting tracking data can be massive in volume. Analyzing these large volumes of tracking data can reveal interesting patterns that can provide opportunities for different types of location-based services, security, indoor navigation, identifying problems in the system, and finally service improvements. In addition to the huge volume, the structure of the unprocessed raw tracking data is complex in nature and not directly suitable for further efficient analysis. It is essential to develop efficient data management techniques and perform different kinds of analysis to make the data beneficial to the end user.

The Ph.D. study is sponsored by the BagTrack Project (http://daisy.aau.dk-/bagtrack). The main technological objective of this project is to build a global IT solution to significantly improve the worldwide aviation baggage handling quality. The Ph.D. study focuses on developing data management techniques for efficient and effective analysis of RFID-based symbolic indoor tracking data, especially for the baggage tracking scenario. First, the thesis describes a carefully designed a data warehouse solution with a relational schema sitting underneath a multidimensional data cube, that can handle the many complexities in the massive non-traditional RFID baggage tracking data. The thesis presents the ETL flow that loads the data warehouse with the appropriate tracking data from the data sources. Second, the thesis presents a methodology for mining risk factors in RFID baggage tracking data. The aim is to find the factors and interesting patterns that are responsible for baggage mishandling. Third, the thesis presents an online risk prediction technique for indoor moving objects. The target is to develop a risk prediction system that can predict the risk of an object in real-time during its operation so that the object can be saved from being mishandled. Fourth, the thesis presents

two graph-based models for constrained and semi-constrained indoor movements, respectively. These models are used for mapping the tracking records into mapping records that represent the entry and exit times of an object at a symbolic location. The mapping records are then used for finding dense locations. Fifth, the thesis presents an efficient indexing technique, called the DLT-Index, for efficiently processing dense location queries as well as point and interval queries. The outcome of the thesis can contribute to the aviation industry for efficiently processing different analytical queries, finding problems in baggage management systems, and improving baggage handling quality. The developed data management techniques also contribute to the spatio-temporal data management and data mining field.

Summary in Danish / Dansk Resumé

Størstedelen af folks liv foregår indendørs, fx på kontorer, universiteter, indkøbs centre, undergrundsbaner stationer, lufthavne, museer, lokalcentre osv. Sådanne lokationer kan være meget store og det kan være svært at finde vej. Udvikling af indendørs sporings teknologier som fx RFID, Bluetooth og Wifi kan spore folk og objekter fra en symbolsk lokation til en anden inden i et indendørs område. Den resulterende sporingsdata kan være meget stor. Analyse af denne store mængde sporingsdata kan afsløre interessante data mønstre der kan give mulighed for forskellige typer of lokationsbestemte servicer, sikkerhed, navigation, problem identificering og service forbedring. Den store data mængde er ikke det eneste problem, strukturen af det uprocesseret sporingsdata er kompleks af natur og den er ikke direkte brugbar til effektiv analyse. Det er derfor essentielt at udvikle effektiv datahåndterings teknikker og udføre data analyse for at gøre dette data brugbart for slutbrugeren.

Dette Ph.d. studie er sponseret af BagTrack projektet (http://daisy.aau.dk-/bagtrack). Det primære tekniske mål af dette projekt er at bygge en global IT løsning der skal signifikant forbedre kvaliteten af luftfart bagage håndtering. Ph.d. studiet fokuserer på udvikling af forskellige data håndterings teknikker for effektiv analyse af RFID baseret symbolsk indendørs sporingsdata, især på baggage håndtering. For det første så designer Ph.d. afhandling en data warehouse løsning baseret på et relationelt schema, oven på dette er der bygget en multidimensional data kube. Denne kube kan håndtere mange af kompleksiteterne fra det massive RFID bagage sporings system. ETL processen indlæser sporingsdataet ind i data warehouset fra datakilderne. For det andet så udvikler dette Ph.d. studie en metodologi for at finde risiko faktorer i RFID bagage sporingsdata. Målet er at finde faktorer og interessante mønstre der er for ansvarlige for baggage fejlbehandling. For det tredje så bliver der i dette studie udviklet en online risiko forudsigelses teknik for objekter der bevæger sig i symbolske lokationer. Den udviklede teknik vil

skulle være i stand til at forudsige om et objekt er i fare i realtid under selve baggage håndteringen. For det fjerde så designes der to modeller baseret på grafer, en for begrænsninger og en anden for semi-begrænset indendørs bevægelser. Modellerne bliver brugt til afbilding af sporings registre til registre der repræsentere ind- og udgangs tider for objekter i symbolske lokationer. Disse afbildinger registre bliver brugt til at finde lokationer med høj objekt tæthed. For det femte så designer denne afhandling en effektiv Indeksering teknik, kaldet DLT-Index, til effektiv processering af lokationer med høj objekt tæthed så vel som punkt og interval forsørgelser. Denne afhandling giver effektiv processering af forskellige analytiske forsørgelse, finder problemer i bagage håndterings systemer, og forbedre bagage håndterings kvaliteten for luftfartsindustrien. De udviklede data håndterings teknikker og analyser bidrager til spatio-temporal data håndtering og data mining forsknings områderne.

Acknowledgements

First of all, I praise the almighty Allah for giving me strength to complete this thesis.

I sincerely like to thank my honorable Ph.D. advisor, Professor Torben Bach Pedersen for giving me the opportunity to work with him. I must have to express that he is an amazing Ph.D. supervisor and I am extremely lucky to be one of his Ph.D. students. My words are powerless to express my gratitude to him. His patient guidance, encouragement, carefulness, advice, and prompt replies to questions and queries have played the important key role in this thesis. He has always provided very concrete ideas and comments throughout my Ph.D. research. I am extremely grateful to him for his enormous support during my Ph.D. study.

I would also like to express my sincere appreciation and my sincere gratitude to my Ph.D. co-advisors Associate Professor Hua Lu and Associate Professor Toon Calders for guiding me and teaching me different technologies during my Ph.D. research. I am very grateful for their valuable contributions to the publications. Their reviews and comments on the articles were very constructive. I have to say that without their help this thesis would not be a reality.

I would like to thank all my colleagues in the Computer Science department at Aalborg University especially the Database and Programming Technologies group. They were always ready to help me whenever I was in need. They were very talent as well as very friendly, and I had really a great time in the group. Special thanks to Kim Ahlstrøm Jakobsen for helping me on writing the Danish summary. Thanks also to administrative staff Helle Schroll and Helle Westmark for their valuable supports regarding administrative issues. I would also like to appreciate the prompt help of IT support staff Lon Nguyen. I also do not want to miss the opportunity to thank my colleague Asif Iqbal Baba to whom I spent a big part of my department life as well as social life during my Ph.D.

I also would like to thank all my colleagues in the WIT research group at Université Libre De Bruxelles where I spent 10 months of my Ph.D. study. Special thanks to Professor Esteban Zimányi for his help regarding many

administrative matters of the IT4BIDC. I also appreciate the help of administrative staff Holly Pharoah and Vinciane De Wilde regarding my stay at ULB.

My sincere gratitude to my teacher Md. Manirul Islam, Assistant Professor of Computer Science Department, American International University-Bangladesh (AIUB), who encouraged me and helped me in many aspects of my life. I also express my appreciation to my teacher Md. Mashiour Rahman, Senior Assistant Professor, AIUB for his help in my academic career. I cannot thank both of them enough for helping me.

I acknowledge the Danish National Advanced Technology Foundation for financial support. This research was supported by the BagTrack project under grant no. 010-2011-1. I also like to thank the project partner Lyngsoe Systems A/S for providing real RFID baggage tracking data set.

Finally, and most importantly I would like to give special thanks to my wife, Nadia Islam and my daughter, Manha Ahmed. I am in debt to them for not giving proper time due to my study. Nadia's love, patience, encouragement, supports, and understanding helped me to work better. Special thanks to my parents whose prayers and supports are always with me. I would also like to thank my parents-in-laws, brothers, sister, friends, and other family members in Bangladesh for their prayers, sacrifice, and supports. Last but not least, I would also like to thank the Bangladeshi people in Aalborg for their supports and activities.

Contents

Thesis Details

Thesis Title:	Analytics on Indoor Moving Objects with Applications in Airport Baggage Tracking
Ph.D. Student:	Tanvir Ahmed
Supervisors:	Prof. Torben Bach Pedersen, Aalborg University (AAU Main Supervisor) Assoc. Prof. Hua Lu, Aalborg University (AAU Co-Supervisor) Assoc. Prof. Toon Calders, Université Libre De Bruxelles (ULB Supervisor)

The main body of this thesis consist of the following papers.

[1] Tanvir Ahmed, Torben Bach Pedersen, Hua Lu, "A Data Warehouse Solution for Analyzing RFID-Based Baggage Tracking Data," *Proceedings of the IEEE 14th International Conference on Mobile Data Management(IEEE MDM 2013), Milan, Italy,*, vol. 1, pp. 283–2929, 2013.

[2] Tanvir Ahmed, Toon Calders, Torben Bach Pedersen, "Mining Risk Factors in RFID Baggage Tracking Data," *Proceedings of the IEEE 16th International Conference on Mobile Data Management(IEEE MDM 2015), Pittsburgh, Pennsylvania, USA*, vol. 1, pp. 235–242, 2015

[3] Tanvir Ahmed, Toon Calders, Torben Bach Pedersen, Hua Lu "Online Risk Prediction for Indoor Moving Objects," *Accepted in the proceedings of the IEEE 17th International Conference on Mobile Data Management(IEEE MDM 2016), Porto, Portugal*, 10 pages, 2016

[4] Tanvir Ahmed, Torben Bach Pedersen, Hua Lu,"Finding Dense Locations in Symbolic Indoor Tracking Data: Modeling, Indexing, and Processing," *Under revision for a Journal publication*, Submitted on June, 2015

In addition to the main papers, the following publications have also been made.

[1] Tanvir Ahmed, Torben Bach Pedersen, Hua Lu, "Capturing Hotspots For Constrained Indoor Movement," *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2013), Orlando, Florida, USA*, pages 462-465, 2013.

[2] Tanvir Ahmed, Torben Bach Pedersen, Hua Lu, "Finding Dense Locations in Indoor Tracking Data," *Proceedings of the IEEE 15th International Conference on Mobile Data Management(IEEE MDM 2014), Brisbane, Australia*, vol. 1, pp. 189-194, 2014

This thesis has been submitted for assessment in partial fulfillment of the joint Ph.D. degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited, DOI pointers and/or copyright/credits are placed prominently in the references.

Chapter 1

Introduction

1 Background and Motivation

A study shows that people spend approximately 87% of their time in indoors, 5-6% in a vehicle, and 7-8% in outdoors [51]. Indoor spaces touch a number of aspects of our life like office and university buildings, airports, subway stations, shopping malls, community centers, museums, and so on. Indoor tracking technologies based on means such as Radio Frequency Identification (RFID) [77], Bluetooth [33], and Wi-Fi [21] can continuously track and record object positions and movements in large indoor spaces. For example, tracking movements of people inside the different parts of a museum and various sections in an airport, tracking of items positions and movements in supply chain management systems, baggage tracking in airports, books tracking in libraries, etc. However, the unprocessed raw tracking data are complex in nature, massive in volume, and rich in low level details, all having impacts on the further analysis. Moreover, as the consequence these problems also badly affect making any strategic decision from the produced data. For example, Walmart recently started using RFID at the item level. The research firm Venture Development Corporation [12] predicts that this will generate up to 7 terabytes of data per day [41]. Managing and querying outdoor tracking data is well established and has been studied over the past decade [39, 66, 78]. However, the indoor tracking scenario significantly differs from the GPS and cellular based outdoor tracking scenarios as geometric representations such as the linear model are not suitable in the indoor setting. In an indoor tracking scenario, the presence of an object at a location is only visible when the object comes under the activation range of a tracking device deployed at that location, whereas, GPS and cellular based outdoor tracking can continuously report the position of objects. Thus, indoor movements are modeled differently as compared to outdoor movements. In the case of indoor movement,

symbolic models [48] are used, whereas, in the outdoor scenario geometric models and spatial networks are used. In general, managing and querying the nontraditional large volume of low-level indoor tracking data is a critical issue for the analyst. The aim of the Ph.D. study is to develop efficient data management techniques to facilitate efficient and effective analysis over the unprocessed large volume indoor tracking data.

The Ph.D. work is being carried out in connection with a major industry initiative called The BagTrack Project [1]. The project partners include IATA, SAS airline, Aalborg Airport, Lyngsoe Systems [6], Department of Mathematical Science, AAU and Center for Data-intensive Systems (Daisy), Department of Computer Science, AAU. The main technological objective of this project is to build a global IT solution to significantly improve the worldwide aviation baggage handling quality. More details about the project can be found at this link: http://daisy.aau.dk/bagtrack.

The baggage reports of SITA [9] disclose the immense loss caused by baggage mishandling in the aviation industry. Every year more than 31M passengers and 34M bags are affected by baggage mishandling. Such a passenger wastes on average 1.7 days of his vacation or business trip waiting for the mishandled bag. Overall, baggage mishandling problems cost a total of 3,300M USD/year to the airline sector. In addition to this enormous loss, the baggage mishandling frustrates the passengers. Therefore, the aviation industry faces a major challenge to solve the problem of mishandled bags.

The project's industry partner Lyngsoe Systems (LS) [6] implements RFID-based baggage tracking systems in a number of Scandinavian airports. LS deploys RFID readers at different baggage handling locations such as check-in, sorter, gateway, belt loader, etc., and uses RFID technology for tracking the movements of bags during their operation in the airports. The tracking results in a large volume of raw RFID baggage tracking data. Due to the large volume, rawness, and lack of efficient structure, the report generation and query processing is extremely slow as well as not suitable for analyzing the reasons of baggage mishandling. So, it requires efficient data management techniques for easy and fast queries on the data set to get more insight into the data and see them from different dimensions. It also requires predictive models that can find interesting patterns and factors from the data set that are closely related to baggage mishandling. Further, dense location extraction for finding over-loaded locations, real-time baggage risk prediction so that a risky bag can be saved, and other kinds of analytics can utilize the generated data for system improvements and location based services, and can create more business opportunities. To solve these issues and facilitate more opportunities, the thesis proposes several novel techniques that are briefly discussed in section 3.

2 Symbolic Indoor Tracking

In an indoor tracking system, tracking devices are strategically deployed at different fixed symbolic locations, such as different doors in an office space, between sections in an airport, different locations in airport baggage management, etc. The objects contain tags or devices that can be tracked by the tracking devices. For example, in the case of RFID technology, RFID readers and RFID tags are used; in the case of Bluetooth systems, Bluetooth access points and Bluetooth devices are used. After deployment of the tracking devices, the positions are recorded in the database.

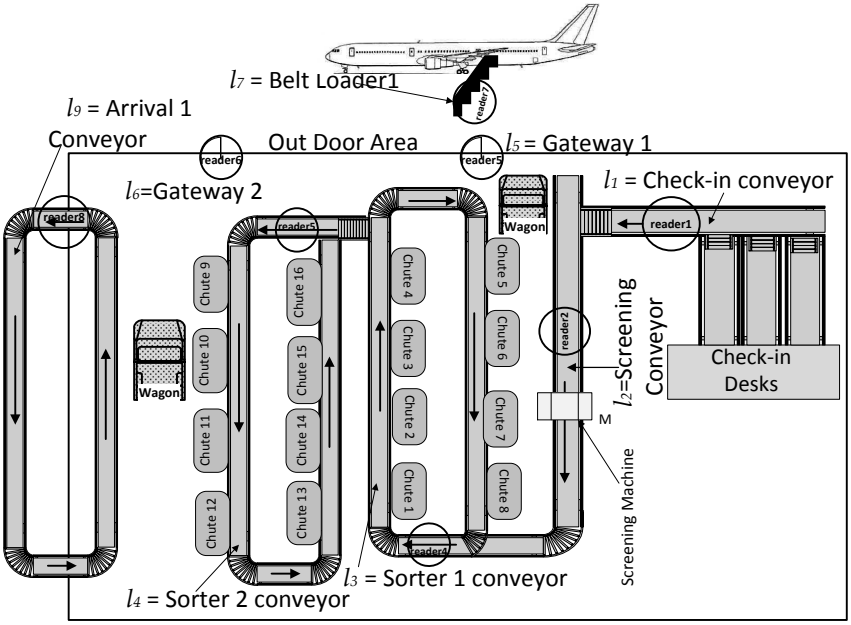


Fig. 1.1: RFID reader deployment for airport baggage tracking

Fig. 1.1 shows an example of RFID baggage tracking scenario. The circles represent the activation ranges of the RFID readers. Fig. 1.2 shows an example of airport baggage flow inside an airport and across multiple airports. The upper part of Fig. 1.2 shows the top level path of a bag that traveling from Aalborg Airport (AAL) to Brussels Airport (BRU) via Copenhagen Airport (CPH). The bag has to go through several baggage processing steps inside each airport. The bottom part of Fig. 1.2 shows the baggage processing stages inside AAL. The circles represent the baggage tracking locations where RFID readers are deployed for baggage tracking. Before handing over a bag into the system, an RFID tag with some encoded information about the

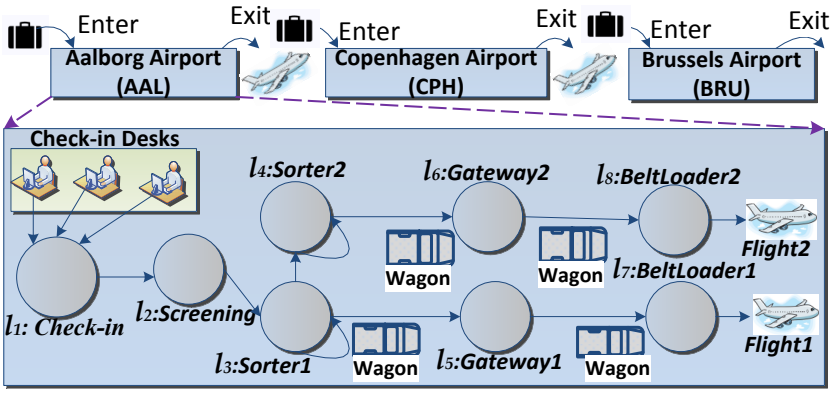


Fig. 1.2: Baggage flow inside an airport and across multiple airports

bag and the route is attached to the bag. Suppose the bag is intended for *Flight1* and the expected path for the bag is: "*check-in* → *Screening*→ *Sorter1* → *Gateway1* → *BeltLoader1*". Mismanagement or inefficiency at any one of these transitions may result in the bag being mishandled, i.e., the bag might miss the flight due to delay, or the bag might be sent to wrong flight. While passing through the different locations, the bag enters the activation range of an RFID reader, it is continuously detected by the reader with a sampling rate, and it generates *raw reading records* with the form: $\langle obj, Loc, t \rangle$. It means that a reader placed at location *Loc* detects a moving object *Obj* in its activation range at time t. An example set of raw reading records in a symbolic indoor tracking system for the scenario of Fig. 1.2 is shown in Table 1.1. It can be seen that there can be several readings when an object go through under the activation range of a reader. As a result, the produced raw reading records are generally huge in volume and not well structured. So, they need to be managed in an efficient way for further analysis.

Table 1.1: Raw Tracking Data

Obj	$\langle$ *Obj, Loc, t* $\rangle$
o_1	$(o_1, l_1, 1)$ $(o_1, l_1, 3)$ $(o_1, l_1, 5)$ $(o_1, l_2, 12)$ $(o_1, l_2, 14)$ $(o_1, l_3, 25)$ $(o_1, l_3, 27)$ ...
o_1	$(o_2, l_1, 10)$ $(o_2, l_1, 12)$ $(o_2, l_2, 26)$ $o_2, l_3, 32)$ $(o_2, l_4, 39)$ $(o_2, l_4, 41)$ $(o_2, l_6, 46)$ $(o_2, l_8, 55)$ ...
...	...
o_{1000}	...

3 Thesis Overview

This section gives an overview of each chapter and the appendices in this thesis.

3.1 Chapter 2: Data Warehouse Solution

A data warehouse is a repository of historical data for the purpose of analysis. Usually multidimensional modeling is applied to the data of the data warehouse to make it usable for analysis and decision making. An OLAP cube uses the multidimensional model and stores summarized and aggregated information. It facilitates rapid and uniform response times to aggregate queries from the large data set [49].

Chapter 2 presents a data warehouse solution for storing and analyzing RFID-based airport baggage handling data. During the travel from the origin airport to the final destination, a bag has to be processed at different locations and has to move through a sequence of locations or steps such as check-in, screening, sorter, gateway, loading, unloading, transfer, arrival, etc. A large number of stakeholders are involved in these steps and inaccuracy or inefficiency at any one of these steps can lead to the bag being mishandled, i.e., the bag may be delayed in the airport and miss the flight or may be sent to a wrong airport. Deployment of RFID readers at different baggage handling steps can help track the baggage movements from one symbolic location to another. The generated huge volume of tracking data needs to be stored in a well-structured way for efficient analysis. Further, incorporating other different kinds of dimensions [65] such as route, airports, airline, handler, transit time, special events, etc., the tracking data can reveal interesting information about the baggage handling quality, finding problems and relevant solutions. The analysis can also indicate new opportunities as well as actions for service improvements. In addition to the relational data warehouse schema, the chapter also builds a multidimensional data cube on top for efficient query processing. The data warehouse supports complex queries to analyze the data from different dimensions with various level of granularity. For example, the manager of Aalborg airport may ask a query like *how many bags originating from Aalborg airport have missed their flights during the Aalborg Carnival of 2015*. Another query can be *find the average number of bags traveling to Asia from Aalborg in the morning of each Monday*. As mentioned above, due to the massive volume, rawness, and other reasons the queries are very slow. Moreover, due to a lack of proper structure, in many cases it is not possible to get more insight into the data and generate reports involving various dimensions. The proposed data warehouse facilitates more insight into the data, enables reports involving various dimensions as well as processing complex queries very efficiently and effectively.

In general, in an RFID tracking setting, the raw reading records are converted into tracking records that represent the first and last times an object stayed under the activation range of an RFID reader [76]. There is some research on RFID data warehousing solutions for supply chain systems [32, 38]. However, the setting in the RFID baggage tracking differs from other kinds of scenarios such as supply chain and logistics, as the locations in baggage tracking are much more fine-grained and objects move fast between locations. Further, baggage stay between locations is more important than stay under a reader's activation range. So, in this chapter, the proposed data warehouse introduces the concept of stay records where a stay record maintains the object transitions between locations. The data warehouse also considers other important dimensions for complex analysis on the baggage tracking data. The designed data warehouse and cube also deal with the complex many-to-many relationship between bag and flight. The chapter also discusses the steps, complexities, and solutions of loading the data warehouse with the appropriate tracking data from the source system. The proposed designs are implemented with several technologies and the query processing times are compared. The result shows that for some queries the cube is 7 times faster compared to a relational data warehouse and 2300 times faster compared to source data. Generally , a nontechnical user accesses and uses a data warehouse and cube using some visualization tools. The chapter also shows that the data warehouse and the cube can be accessed and visualized through a very user friendly business intelligence tool called Targit BI suite [11]. The proposed data warehousing concepts can be generalized for multi-site based symbolic indoor, outdoor, and mixed indoor-outdoor tracking systems.

3.2 Chapter 3: Mining Risk Factors

Data mining is the process of extracting interesting patterns and knowledge from large data sets [42]. In the baggage handling scenario, mining the baggage handling data can disclose interesting patterns and reasons of baggage mishandling. Chapter 3 presents a detailed methodology for mining risk factors in RFID baggage handling data. For example, from the baggage handling data, we may be interested in knowing, how the baggage mishandling is related to the transit time, operating airport, day of week, and time in a day and it might results in a rule that can disclose the probability that a bag will be mishandled if it has 40 minutes transit time at Copenhagen airport on Monday morning.

Before performing data mining, the data have to be well prepared such that the gained knowledge is useful. As the baggage tracking data are at low level, it requires some preprocessing to extract important and relevant features before performing further analysis. Additionally, the baggage tracking data are generally highly imbalanced as the mishandling rate is very low

compared to correctly handled bags. In our experimental data set, the mishandling rate is only 0.8%. Most of the data mining techniques suffer from such imbalance problem in the data set. This makes the mining process biased towards predicting that all bags will be handled correctly by default and makes it difficult to learn rules related to mishandled bags. Thus, in addition to the preprocessing for feature extraction, special care has to be taken to deal with this class imbalance problem to get patterns of higher quality.

Warehousing and mining techniques of RFID item tracking data in supply chain systems have been proposed in [41]. Like in other related work, they also convert the raw RFID records into cleansed records containing the first and last reading times of an object under the readers' activation range. As discussed in the previous section, in our data warehouse we introduce *stay records*. In the case of data mining, we further refine the *stay records* into *FlightLeg records* that capture different aggregated information from the stay records including other dimensional information for a higher-level analysis. While traveling from origin to final destination, a bag may have to take multiple flights. The journey between the source airport to the destination airport for each flight is known as a leg for the entire journey of the bag, i.e., a bag taking two flights has two legs. A *FlightLeg record* of a bag includes the from and to airports, transfer status of the bag, weekday, hour of the flight time, available duration before the flight, status indicating whether a bag took longer between any pair of locations, duration of the delay in arriving the flight at the transfer airport, total number of bags processed during the flight hour, and final status of the bag, i.e., mishandled or not at that leg. In the *FlightLeg records*, we maintain one record for each leg of a bag for its journey from origin to final destination.

The problem of mining imbalanced data sets has been addressed frequently in research [17, 28]. Mostly, the imbalance problem is solved by re-sampling that includes under-sampling and over-sampling. Another approach is cost-sensitive learning. In our approach, we apply several data mining techniques such as *Decision Tree (DT) [71], Naive Bayes classifier (NB) [25], KNN classifier (KNN) [25], Linear regression (LIR) [4], Logistics regression (LOR)* [5], and *Support vector machine (SVM)* [67] with different re-balancing techniques such as under sampling and over sampling. Then, based on the area under the curve (AUC) of ROC and the precision-recall (PR) curve we find out which data mining techniques can provide us with the best models.

Before mining, we fragment the data set based on transit status (i.e., whether bag is in the transfer airport or not) and available transfer time during the transit as most of the mishandling occurs during transfer. We performed a comprehensive experiment and it shows that the proposed methodology can reveal interesting insights into the data set. The extracted patterns show that bags are generally mishandled for shorter available processing time before the flight. A bag is considered to be at high risk if it has less than 54

minutes to catch the next flight at the transit airport. Further, the results also show that different airports act differently for non-transfer bags. Moreover, a longer stay between baggage handling locations and the busyness of the airport are also important factors for mishandling bags.

3.3 Chapter 4: Online Risk Prediction

The proposed data warehouse mainly facilitates exploratory data analysis, whereas, the proposed mining methodology can build prediction model and automatically extract interesting patterns and reasons for baggage mishandling. The outcomes from the data warehouse solution and the data mining methodology discussed in Chapter 2 and Chapter 3, respectively, can lead to taking actionable decisions for improving baggage handling quality and other useful services. However, in a time critical application such as airport baggage tracking, predicting the risk of a bag in real-time during their operation at the airport can help to take immediate action so that the bag can be saved from being mishandled. Chapter 4 presents a novel probabilistic approach for online risk prediction for indoor moving objects, especially for the baggage handling scenario.

In existing literature, mainly data mining is performed on the spatio-temporal indoor and mixed indoor-outdoor tracking data for finding spatiotemporal patterns, frequent and typical movements and walks [26, 69]. Further, they are mainly focused on the offline scenario. The chapter addresses a new perspective of indoor tracking data analysis, which is the online risk prediction for indoor moving objects. An example of a real-time request can be: *"notify the baggage management team whenever a bag becomes risky during its processing time at Copenhagen airport"*. Another request can be: *"which are the ten current bags with the highest risk of not reaching their plane on time?"* To answer these kinds of requests, the chapter proposes a probabilistic flow graph (PFG) and an aggregated probabilistic flow graph (APFG) that capture the historical object transitions and the durations of the transitions. The probabilistic information is stored in a set of histograms called least duration probability histogram (LDPH) and aggregated LDPH (ALDPH). The histograms are annotated to the edges of the graphs. Then the graphs are used for getting a risk score of an online object and uses the score for predicting the riskiness of the object. After building the models, a risk score threshold is optimized for each of the pre-planned path sequences in the data set. The chapter presents a cost model for obtaining the thresholds that maximize the overall benefit of identifying and handling the predicted risky objects. The thresholds are used for computing the maximum time limit of an object for a particular transition. The online risk prediction system uses a time trigger that fires when an object crosses the time limit. The proposed prediction system also considers the available processing time of an object

(e.g., available processing duration of a bag before its flight) for risk prediction that results in specialized risk score thresholds as well as time triggers for each object. This specialization helps to predict the riskiness of an object as early as possible as well as reduces false alarms.

The chapter reports a comprehensive experimental study with multiple synthetic data sets following different distributions and a real baggage tracking data set. The results show that the prediction method can identify the risky objects very accurately when the objects approach the bottleneck location in their path. The methods work very well in a class imbalanced dataset. Further, the risky objects are predicted well in advance such that they can be saved from being mishandled. The results also show that the prediction system can save 83.4% of the total cost in the case of our experimental data set.

3.4 Chapter 5: Dense Location Extraction

Extracting the dense locations in large indoor spaces can benefit many applications by obtaining overloaded areas, for security control, crowd management, indoor navigation, and so on. As discussed earlier, indoor tracking data can be enormous. Further, they are not immediately ready for finding dense locations. Chapter 5 presents some efficient data management techniques for finding dense locations in symbolic indoor space.

There is research that addresses density queries and hot route queries on road networks [52, 55]. However, it has been shown that the geometric polyline representation for outdoor trajectories is unsuitable for indoor trajectories [47]. For example, consider an RFID tracking application where RFID readers are deployed at the doors of different rooms in a large indoor space. Each reader has a very limited tracking range that covers a small portion of the room, e.g., only the door of a room. If an object containing an RFID tag moves from one room to another, this produces two consecutive tracking records of the object location in different rooms. Due to limitations in indoor positioning technologies, the locations of the object between these two records are not obtained. Moreover, in a constrained indoor movement, such as baggage tracking, a reader is deployed at any point of a conveyor belt or baggage handling location. So, it requires a mapping technique to obtain the times when an object entered and exited a location. Furthermore, the generated large volumes of data require efficient query processing techniques. The chapter takes all these challenges into account and proposes an efficient approach for finding dense locations in indoor tracking data.

Based on the path inside a location, the chapter divides the indoor spaces into two categories. One is constrained path space, e.g., baggage handling path in the conveyors of an airport, and another one is semi-constrained path space, e.g., rooms in a building, different sections in an airport where peo-

ple move from one section to another. Then the chapter proposes two graph models for each kind of indoor spaces, respectively. The models are used for mapping the indoor tracking records into mapping records, where each mapping record represents the entry and exit times of an object in symbolic locations. Then the mapping records are used for finding dense locations. The chapter proposes a new efficient index structure called the *Dense Location Time Index* (*DLT*-Index) that stores aggregate information, i.e., the number of (unique) objects entering, exiting, and present at a location at different timestamps or time intervals. The chapter also presents some efficient data processing techniques, including algorithms for *DLT*-Index construction and updating, data pruning methods in the *DLT*-Index, and algorithms for finding dense locations using the *DLT*-Index. The proposed index is also used for efficient processing of aggregate *point, interval*, and *duration queries*. The *DLT*-Index is generalized such that it can be used for indexing any type of time intervals and it enables us to query for the distinct number of records at a given time point, as well as for a given time interval. The chapter reports a comprehensive experimental evaluation using both real and synthetic data. The results show that the proposed solution is efficient and scalable. The proposed *DLT*-Index can process the dense location queries very efficiently. The results also show that the *DLT*-Index can process point queries 340 times faster and interval queries 300 times faster compared to SQL in RDBMS.

3.5 Appendices A and B

Appendices A and B do not contain any new content compared to Chapter 5. However, they are included in the thesis for completeness, as they have been published as separate papers. Appendix A contains the graph model for constrained indoor movements and a naive approach of dense location extraction. Appendix B mainly presents the graph model for semi-constrained indoor movements, the *DLT*-Index, and query processing using the *DLT*-Index.

4 Structure of the Thesis

The thesis is organized as a collection of individual papers. Each chapter/appendix is self-contained and can be read in isolation. There can be some overlaps of concepts, examples, and texts in the introduction and preliminaries sections of different chapters as they are formulated in relatively similar kind of settings. The chapters have been slightly modified during the integration such that, for example, their bibliographies have been combined into one, and references to "this paper" have been changed to references to "this chapter". Appendix A and B are the two published conference papers that

are combined and extended for a journal paper which is presented in Chapter 5. So, Appendix A and B do not contain any new content compared to Chapter 5.

The papers included in this thesis are listed in the following. Chapter 2 is based on Paper 1, Chapter 3 is based on Paper 2, Chapter 4 is based on Paper 3, Chapter 5 is based on Paper 4, Appendix A is based on Paper 5 and Appendix B is based on Paper 6. As mentioned earlier, Paper 4 is an extended version of Paper 5 and Paper 6.

1. T. Ahmed, T. B. Pedersen, and H. Lu. A data warehouse solution for analyzing RFID-based baggage tracking. In *14th IEEE International Conference on Mobile Data Management (IEEE MDM 2013), Milan, Italy*, pages 283–292, 2013.

2. T. Ahmed, T. Calders, and T. B. Pedersen. Mining risk factors in RFID baggage tracking data. In *16th IEEE International Conference on Mobile Data Management (IEEE MDM 2015), Pittsburgh, Pennsylvania, USA*, pages 235–242, 2015.

3. T. Ahmed, T. B. Pedersen, T. Calders, and H. Lu. Online Risk Prediction for Indoor Moving Objects. *Accepted in the proceedings of the IEEE 17th International Conference on Mobile Data Management(IEEE MDM 2016), Porto, Portugal*, 10 pages, 2016

4. T. Ahmed, T. B. Pedersen, and H. Lu. Finding Dense Locations in Symbolic Indoor Tracking Data: Modeling, Indexing, and Processing. Under revision for a publication in *Geoinformatica*, submitted on 25-June-2015, 39 pages.

5. T. Ahmed, T. B. Pedersen, and H. Lu. Capturing Hotspots For Constrained Indoor Movement. In *21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2013), Orlando, Florida, USA*, pages 462–465, 2013.

6. T. Ahmed, T. B. Pedersen, and H. Lu. Finding Dense Locations in Indoor Tracking Data. In *15th IEEE International Conference on Mobile Data Management (IEEE MDM 2014), Brisbane, Australia*, pages 189–194, 2014.

Chapter 2

A Data Warehouse Solution for Analyzing RFID-Based Baggage Tracking Data

The paper has been published in the
Proceedings of the IEEE 14th International Conference on Mobile Data Management(IEEE MDM 2013), Milan, Italy, pp. 283–292, 2013. The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1109/MDM.2013.42

Abstract

Today, airport baggage handling is far from perfect. Baggage goes on the wrong flights, is left behind, or gets lost, which costs a lot of money for the airlines, as well as frustration for the passengers. To remedy the situation, we present a data warehouse (DW) solution for storing and analyzing spatio-temporal Radio Frequency Identification (RFID) baggage tracking data. Analysis of this data can yield interesting results on baggage flow, the causes of baggage mishandling, and the parties responsible for the mishandling(airline, airport, handler,...), which can ultimately lead to improved baggage handling quality. The chapter presents a carefully designed data warehouse (DW), with a relational schema sitting underneath a multidimensional data cube, that can handle the many complexities in the data. The chapter also discusses the

Extract-Transform-Load (ETL) flow that loads the data warehouse with the appropriate tracking data from the data sources. The presented concepts are generalizable to other types of multi-site indoor tracking systems based on Bluetooth and RFID. The system has been tested with large amount of real-world RFID-based baggage tracking data from a major industry initiative. The developed solution is shown to both reveal interesting insights as well as being several orders of magnitude faster than computing the results directly on the data sources.

1 Introduction

A recent report[1] discloses the enormous loss caused by baggage mishandling in the aviation industry. Each year more than 31M passengers and 34M bags are affected by baggage mishandling which costs the aviation industry 3,300 M USD. A passenger wastes on average 1.7 days of his vacation or business trip waiting for the mishandled bag. Typical baggage mishandling problems include flight delay, bag loss, wrong bag destination, failure to load bags at the origin airport, missed connection at transit hubs, etc. During the travel from the origin airport to the final destination, a bag is moved over different places in multiple steps: check-in, screening, sortation, loading, transition, arrival, etc. In these steps, a bag is handed over between a large number of stakeholders. In contrast with the visibility of passengers' movement steps during the end-to-end journey, the baggage movement is considerably less visible due to the very limited information to passengers. A single error or inefficacy in a handover can cause a bag not to reach its intended destination with its owner [75].

Radio Frequency Identification (RFID) technology is used in many applications for monitoring object movement. The use of RFID in baggage tracking systems enables to track a bag in an airport as well as along its travel route cross airports. RFID tracking systems generate huge amounts of data. For example, Walmart has recently started using tags at the item level and research firm Venture Development Corporation predicts that this will generate up to 7 terabytes of data per day [38]. In RFID-based airport baggage tracking systems, these huge amounts of RFID data can be very useful for analyzing and mining purposes. Coupled with other kinds of information about routes, transit airports, transit durations, flights and punctuality, airlines, handler, special events, etc., RFID baggage tracking data can reveal a lot of information about baggage handling quality. Analyzing these data will open the door to identifying the different problems in baggage handling and finding solutions for the problems. This is exactly the goal of the BagTrack project (`www.daisy.aau.dk/bagtrack`), within which this work took place. Here, a number of important industry players have teamed up with our data management team to revolutionize baggage handling using RFID.

In this chapter, we present a multidimensional database warehouse solution for RFID-based baggage tracking data. To the best of our knowledge, this chapter is the first to design a multidimensional data warehouse, including a relational DW schema with a data cube on top, for this important domain. The proposed data warehouse contributes to the airline baggage handling process by providing a framework for data analysis and answering complex queries that can ultimately improve the baggage handling quality.

[1] SITA Baggage Report 2012 `www.sita.aero/content/baggage-report-2012`

For example, the manager of Copenhagen airport may ask a query like *how many bags were sent to wrong destination from Copenhagen airport in the Easter holidays of 2012*. Another query can be *find the average number of baggage traveling from Copenhagen airport in the afternoon of each Sunday*. Our data warehouse supports such useful queries effectively and efficiently.

Our data warehouse design features several novelties. First, it captures not only the RFID baggage tracking data but also baggage flow along different dimensions like airport, airline, date and time, all at several levels of granularity. We also handle the complex many-to-many relationship between bag and flight effectively. Second, our design treats date and time as different dimensions, each with different hierarchical levels. Each date is *localized* to a particular airport, to allow capturing special local events, like strikes, sports games, etc. This localization makes it easy to ask complex queries about baggage movement in a particular event or occasion. Third, our data warehouse design supports powerful data analysis queries in a both easy and efficient way. The relevant results enable the users (e.g., an airline) to find out the places (e.g., a particular airport) where mishandling occurs most of the time, and thus to determine the reasons of baggage mishandling. The chapter also discusses the complexities of performing Extract-Transform-Load (ETL) to load the DW from the source data, including our solutions to the encountered challenges.

The remainder of the chapter is organized as follows. Section 2 reviews related work. Section 3 presents an overview of RFID-based airport baggage handling process and the structure of RFID data. Section 4 presents the relational data warehouse design and the multidimensional data cube design. Section 5 represents the ETL steps to load the data from source schema into the data warehouse. Section 6 presents the experimental results. Finally, Section 7 concludes the chapter and points to the challenges for future work that we encountered.

2 Related Work

Data warehousing, mining and work flow analysis techniques have been proposed for RFID-based supply chain systems [36–38, 41]. Those proposals take advantage of bulky movement of objects in supply chains to compress the massive RFID data. Efficient storage scheme and encoding the object paths are designed [53, 54] to support faster query performance for supply chain RFID data. RFID data warehousing for tracking patients and drugs has been studied [34]. The general challenges and solutions for RFID data management are also discussed [27, 76].

The scenario of airport baggage management in this chapter differs from the settings of previous research. Unlike objects in supply chain manage-

ment, airport bags do not move in a bulky way. In addition, the bag locations in airport baggage management are much more fine grained so that places and reasons of mishandling can be identified at a satisfactory level. Further, the time dimension in airport baggage management is also much more fine grained so that complex time-dependent queries can be supported. Last but not least, our data warehouse design considers other important dimensions which enable complex analysis on baggage tracking data.

3 RFID-Based Airport Baggage Handling

In airport baggage management a bag needs to pass different stages to go from origin to final destination. Suppose that Lisa needs to travel from Aalborg Airport (AAL) to Arlanda Airport (ARN) via Copenhagen Airport (CPH). First, Lisa has to check-in and handover her bag to the check-in desk staff. Then the staff puts the bag into the conveyor belt for automatic baggage sortation system. After passing all the stages inside AAL, the bag is loaded into the aircraft using belt loader for the targeted flight. As the bag has to be transferred to ARN, upon arrival at CPH it is shifted to the transfer system. After all the required stages at CPH, the bag is loaded to the aircraft for its next flight to ARN. After arriving at ARN, the bag is shifted to arrival belt and finally Lisa collects the bag from the arrival belt.

Figure 2.1 shows an example of RFID reader deployment at different locations of a baggage management system. An RFID reader corresponds to the location where it is deployed. For example, *reader1* in Figure 2.1 corresponds to *check-in1*, *reader6* corresponds to *Gateway-1* etc. The circles represent the RFID reader and their activation range.

At check-in, an RFID tag is attached to the bag. The tag is then read by readers is passes through their respective activation ranges. An RFID reader continuously detects the tags, and the reader's controller software determines which actual records are stored: if the read rate is 1 sec, an object staying under a reader for 30 sec will generate 30 records, unless it is specified that only one record can be sent per 15 sec, which will yield only 2 records.

An RFID tag has small built-in memory that stores bag information.An example of this data is *{0123456789, 28APR, AAL, SK1234, 28APR, CPH, LH2345, 28APR, ARN}*. The first 10 digits are the *LicensePlate*[2]. Specifically, the 1st digit is a flag, the 2-4th digits state the bag issuer code (e.g., 117 for SK (SAS)) and the 5-10th digits are the baggage tag number. Next, the flight date is 28APR, the tag is printed in AAL(borg), the first flight leg is SK1234 on 28APR to CPH, the second leg is LH2345 on 28APR to ARN.

[2]The license plate is a unique 10 digit code encoding bag information. IATA specifies the rules for using the (Baggage) License Plate in Resolution 740B of their Passenger Services Conference Resolutions Manual.

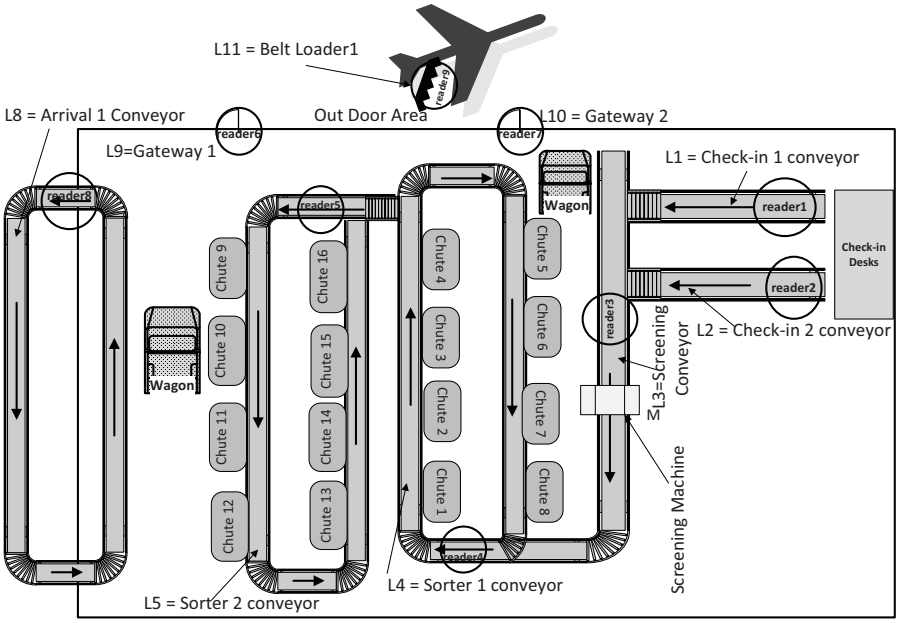


Fig. 2.1: RFID reader deployment for airport baggage handling

Due to the huge number of taggings, the 10 digit integer *LicensePlate* is reused after some time. Thus, a *BagID* is added to uniquely identify a bag. The records are stored in the format: $\langle BagID, L, T, info \rangle$, meaning that a reader at location L detects a bag with ID *BagID* at timestamp T and the tag stores the information *info*. Considering only location and time related information, some example tracking records are shown in Table 2.1. The records are represented in the form: $\langle ReadingID, BagID, LocationID, ReadingTime \rangle$.

Table 2.1: Raw Baggage Tracking RFID Data

⟨ ***ReadingID, BagID, LocationID, ReadingTime*** ⟩
(r1, b1, L1, 1) (r2, b2, L2, 2) (r3, b1, L3, 4) (r4, b1, L3, 5) (r5, b2, L3, 5) (r6, b2, L3, 6) (r7, b1, L4, 8) (r8, b1, L4, 9) (r9, b2, L4, 9) (r10, b2, L4, 10) (r11, b2, L5, 14) (r12, b2, L5, 15) (r13, b1, L4, 19) (r14, b1, L4, 20) . . .

As explained earlier, the raw readings contain many redundant records. A *LocationTrace* table can be constructed from the raw tracking sequence after eliminating the multiple readings. The format of the records in *LocationTrace* table is: *(recordID, BagID, LocationID,* t_{in}, t_{out}*)*, where *recordID* is the identifier

of each location trace record and t_{in}, t_{out} respectively represent the timestamps of first reading and last reading of *BagID* by the RFID reader deployed at location *LocationID*. This means that the bag was within the readers activation range during time t_{in}, t_{out}. An example of a table containing location trace records from Table 2.1 is shown in Table 2.2. In this table the record rec_3 represents, a bag $b1$ is observed by RFID reader at location $L3$ from time t_4 to time t_5, and record rec_5 means that $b1$ is observed by reader at location $L4$ from time t_8 to t_9. This is a lossless compression with huge data reduction in volume.

Table 2.2: Tracking records after duplicate elimination

ObjectID	***TrackingRecord*⟨*RecordID,ObjectID, LocationID*, t_{in}, t_{out}⟩**
b1	*(rec1, b1, L1, 1, 1) (rec3, b1, L3, 4, 5) (rec5, b1, L4, 8, 9) (rec8, b1, L4, 19, 20)*
b2	*(rec2, b2, L2, 2, 2) (rec4, b2, L3, 5, 6) (rec6, b2, L4, 9, 10) (rec7, b2, L5, 14, 15)*
...	...

4 Data Warehouse Design

To support business intelligence analysis using complex analytical queries, we propose a data warehouse design, shown in Figure 2.2. The proposed multi-dimensional data warehouse schema is a mixture of a star and a snowflake schema with one fact table and eight dimension tables. A star schema is less complex compared to a snow-flake schema where the dimensions are partly normalized, but in some places, a snow-flake schema is needed, as discussed in the following. In the following, the different dimensions are described, after which the fact table is introduced to link all the dimensions to the facts and calculated measures. As seen in the Figure 2.2, all the dimension tables follows the standard data warehouse convention of having auto-generated surrogate keys.

4.1 Dimension Descriptions

The different dimensions of the data warehouse presented in Figure 2.2 are described in this section.

Date and TimeOfDay Dimension. As seen in Figure 2.2, date and time has been split up into two dimensions in order to save records compared to a combined dimension table. If date and time were modeled in one table there could be over 86400 records for each day. In a year that would give over 31 million records in the dimension which could result in a slow query

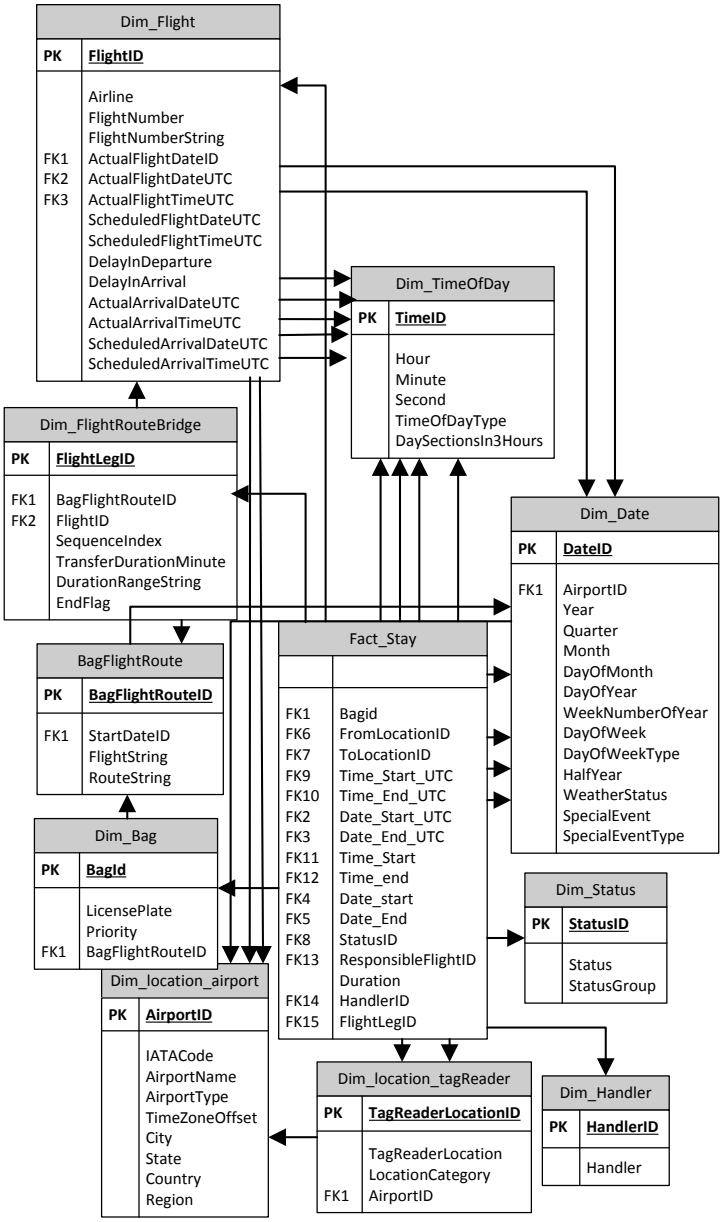


Fig. 2.2: Relational Data Warehouse Schema

performance. Another reason for splitting up the two dimensions is that queries are normally performed on either a date basis or a time of day basis.

The main hierarchy of the *Date* dimension consists of: *Year, Month, and*

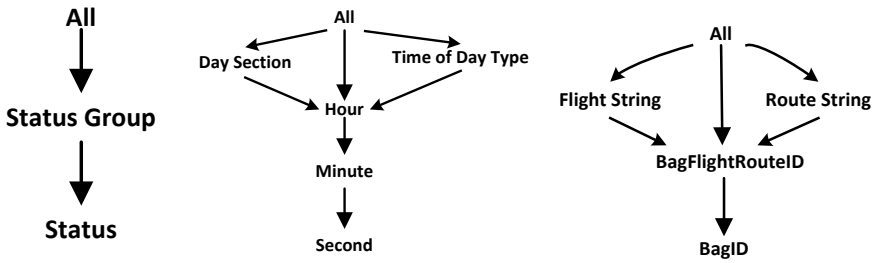
DayOfMonth. To allow more detailed analysis, the dimension includes the following attributes: *HalfYear, Quarter, WeekNumberOfYear and DayOfYear*. *DayOfWeek* is included as a one level hierarchy to specify the specific days of the week. The *HalfYear* attribute indicates if it is the first or last half of the year. The *DayOfYear* attribute enables queries on a specific day or range of days (1..365). The *WeekNumberOfYear* attribute enables queries on specific weeks of a year. The *DayOfWeek* makes it possible to query on a specific weekday and the last attribute *DayOfWeekType* enables classification of days, e.g. holiday, weekend, weekday, etc.

Location based events may occur on a particular date. For example Aalborg, Denmark celebrates a special occasion called Aalborg Carnival which is one of the biggest carnivals in Europe. As a result lot of people from Europe come to Aalborg to celebrate this occasion. Another example can be a strike in the city of an airport or conveyor belt broken of an airport etc. Additionally, bad weather in the area of an airport should be captured to relate baggage handling quality with the weather. So, it is very important to *localize* each date to the airport for capturing these types of location based special events and weather status. We use two special attributes: *SpecialEvent, EventType* which allow queries on a specific event or type of event. For localizing each date and date related information to a particular airport, we use *AirportID* as an attribute of the *Date* dimension. As a result the *Date* dimension has to store a copy of same date for each airport. Figure 2.4b shows the hierarchy of the *Date* dimension.

The *Date* dimension stores a copy of all stored dates for UTC which is independent of the airport. We store a default *AirportID, (AllAirports)*, which indicates all airports and the other attributes also store default values for the corresponding UTC date. An example of date localization concept is shown in Fig. 2.5a. From Fig. 2.5a we can see that date May, 29, 2012 is localized for each airport and as a result it is possible to capture an special event i.e., *ConveyorBeltBroken* at AAL airport.

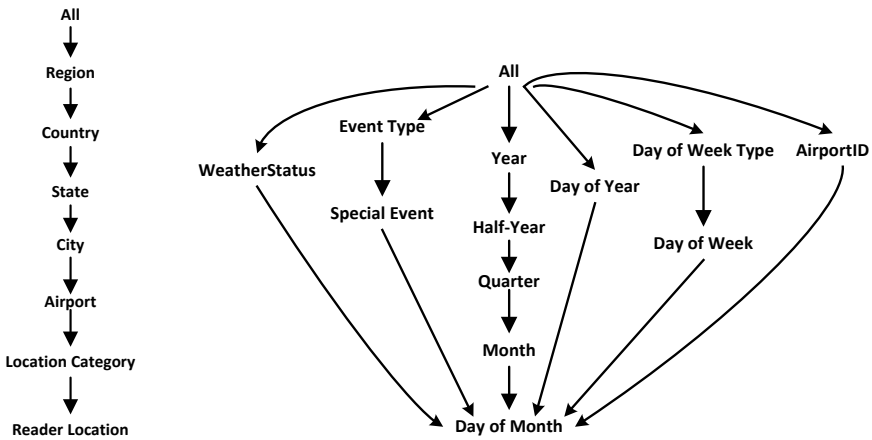
The *Time* dimension called *TimeOfDay* consists of the three necessary attributes to specify a time hierarchy down to a specific second: *Hour, Minute, and Second*. Attribute *TimeOfDayType* indicates whether this is rush hour or normal hour, and *DaySections3Hours* divides the 24 hours of a day into eight 3-hour sections. As a result it will be easier to analyze the baggage movement at different parts of the day. Figure 2.3b shows the hierarchy of the *TimeOfDay* dimension.

Location Dimension. In order to refer to different points of interest in the airport, a hierarchy of seven levels is introduced to handle the various degrees of detail. The lowest level in the hierarchy is denoted as a *TagReaderLocation*, which describes the specific location of an RFID reader like Check-in-1, Sorter-1, Sorter-2, Gateway-1, Arrival-1 etc. A location belongs to a category e.g., Sorter-1, Sorter-2 belongs to the location category Sorter. A number of



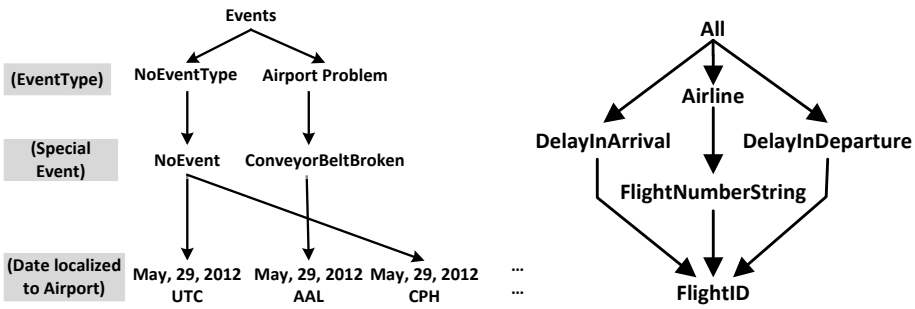
(a) Status **(b)** Time of Day **(c)** Bag

Fig. 2.3: Hierarchy of Status, Time of Day and Bag



(a) Location **(b)** Date

Fig. 2.4: Hierarchy of Location and Date



(a) Date localization **(b)** Flight

Fig. 2.5: Date localization example and Flight hierarchy

location categories are grouped into an airport. The other levels of hierarchy of *Location* dimension are showed in Figure 2.4a.

The location hierarchy enables queries for the baggage movement at different abstraction levels, e.g., from one tag reader location to another tag reader location at the lowest level, and from one region to another region at the top level. Here, the meaning of region indicates the baggage movements between North America, Europe, Middle East and Africa (EMEA), Latin America (LATAM) and Asia-Pacific etc,. The *Location* dimension is snow-flaked at airport level and we use *Dim_airport* as an outrigger dimension [49] as some other dimensions like *Dim_Date, Dim_Flight* etc., depend on the location at the airport level not at the tag reader level.

Status Dimension. The *Status* dimension captures the status of a bag while moving from one location to another. The status domain is {*WrongDestination, OK, LeftBehind, TrackedInNonRouteAirport, StillTrackedInNonRouteAirport, LongerDurationThanExpected, UnexpectedReader*}. To make a hierarchy, another column *StatusGroup* is introduced which contain the group of the status, e.g., *WrongDestination, LeftBehind, TrackedInNonRouteAirport* belong to *mishandled group* etc. Figure 2.3a shows the hierarchy of *Status* dimension.

Bag Dimension. The *Bag* dimension contains the identifier of the bag, *LicensePlate* and other planned travel information of bags like route, airline, date of departure etc. To keep track of the bags planned to move together in the same route and the same flight on the same date, we create a separate ID called *BagFlightRouteID*. This ID is generated while loading the data from the source schema. As a result, it creates a hierarchy in the *Bag* dimension. Each *BagFlightRouteID* corresponds to a given combination of *StartDateID, FlightString* and *RouteString*. Here, *StartDateID* is the departure date, *FlightString* contains a sequence of flights, e.g., "SK1202-QI1354-#"; *RouteString* contains a sequence of airports of the route, e.g., "AAL-CPH-ARN-#". In both strings '#' indicates the end of the sequence. Use of these strings enables queries on flight and route sequences more easily. For example, to find the number of mishandled bags starting from AAL and ending at ARN, the route string wild-card "AAL%ARN:#" can be used. Figure 2.3c represents the hierarchy of *Bag* dimension.

Flight Dimension. In addition to the general information of a flight like *FlightID, Airline, FlightNumber, departure date* and *time* etc., we also store the *source airport, destination airport, delay in departure, delay in arrival etc.* Storing this delay information enables queries that can give an idea about how baggage mishandling is related with punctuality of flights. We store *FlightNumberString* which includes airline and flight number together e.g., *"SK1202"* indicates a flight of SAS (SK) Airline with flight number 1202. Instead of querying only on a flight number it is common to query on a flight based on *FlightNumberString* which can be rolled up to the airline level. Hierarchy of *Flight* dimension is represented in Figure 2.5b.

Handler Dimension. While traveling from origin to destination a bag is handled by many handler organizations at different stages of baggage movement. So handler is an extremely important entity related to baggage management. The *Dim_Handler* table represents the *Handler* dimension. It contains attributes *HandlerID* and *Handler*. The *Handler* dimension enables to find relationship between baggage handling quality and handlers.

4.2 Many-to-Many Relationship Between Flight and Bag

A bag can travel in many flights and a flight can carry many bags, and therefore the relationship between bag and flight is many-to-many. Additionally a sequence number is maintained when a bag is planned for traveling more than one flight. It also includes a transit duration i.e., duration between scheduled departure time of next flight and actual arrival time of the current flight. If a flight is the final flight of a bag in its sequence of flights then the transit duration is remain null. A standard approach of handling many-to-many relationship is the use of a bridge table that use foreign keys from both entities [50] e.g., *Bag-Flight-Bridge*$\langle$*BagID, FlightID, SequenceIndex, TransitDuration*$\rangle$ where *BagID* and *FlightID* are foreign keys taken from *Bag* table and *Flight* table respectively.

However this type of implementation produces huge amount of data in the bridge table which results a low query performance due to join lot of data. In Figure 2.6 the *bag-flight bridge* table shows some examples of association between bags and flights. Even though bags *B1, B2* and *B6* follows same flights with same sequences, due to the database design they have to be inserted thrice. Moreover in the whole table there are 2 different collection of pairs for $\langle$*FlightID, Sequence*$\rangle$ i.e., $\{\langle F1, 1\rangle, \langle F1, 2\rangle\} \rightarrow 1$ and $\{\langle F1, 1\rangle\} \rightarrow 2$. The process of transformation this table is exemplified in Figure 2.6.

Specifically, we create a separate table called *BagFlightRoute* table. It contains an ID *BagFlightRouteID* for each distinct set of pairs. Also, we store the *flight string* and *route string* for each *BagFlightRouteID*. The features of these two strings are described in the earlier sub section. For each *BagFlightRouteID* the pairs and its related info is stored in the *FlightRouteBridge* table. Finally in the *bag* table each bag is assigned with its relevant *BagFlightRouteID*. It makes a one-to-many relationship between *BagFlightRoute* table and *bag* table. The discussed transformation reduces huge amount of rows which results better query performance.

4.3 Fact Table

The *Fact_Stay* is the fact table and it binds the tracking records to the relevant entries in the dimensions. The relations to the *TimeOfDay* and *Date* dimension are represented twice for each *datetime* attribute from the source data. This

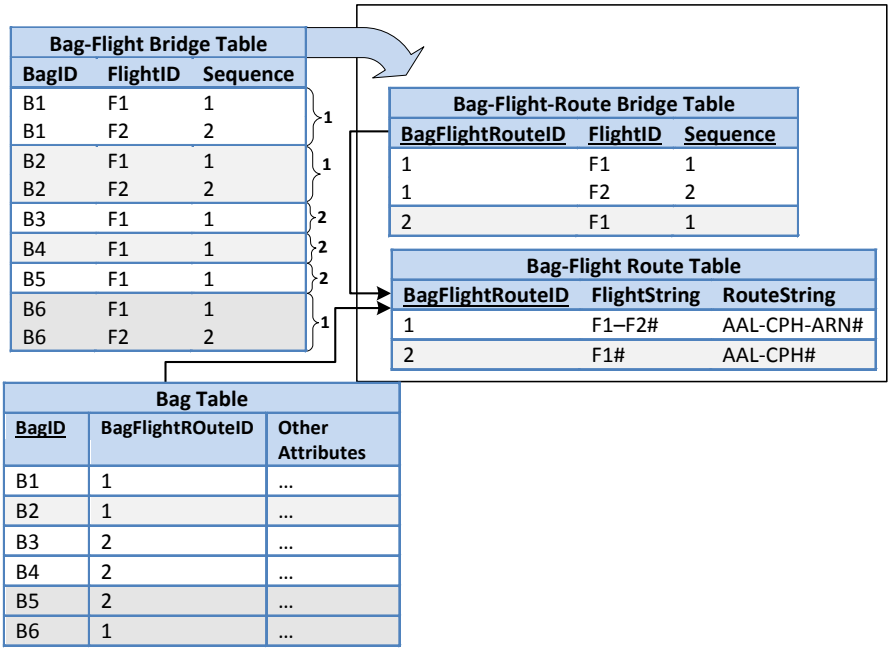


Fig. 2.6: Example of bag-flight many-to-many relationship

is to store both the wall clock time, and the UTC time when considering tracking records from different sites, e.g. AAL where the standard time zone is UTC/GMT+1 hour, Los Angeles International Airport (LAX) where the standard time zone is UTC/GMT-9 hours etc. As an RFID reader does not cover the whole area of a location, the value of t_{out} in Table 2.2 does not necessarily give us the time when an object actually get out from a location. As in the airport baggage tracking scenario a bag moves from one symbolic location to another symbolic location sequentially, instead of storing this t_{out} we store the $time_{end}$ in the fact table which specify when the object reaches another location.

Attributes *FromLocation* and *ToLocation* specify the baggage movement from one tag reader location to the next tag reader location in its path sequence. Usually *time_start* and *time_end* of a particular bag indicates the $time_{in}$ at *FromLocation* and $time_{in}$ at *ToLocation*, respectively. However for the final location (which means there is no reading of a bag after this location), *FromLocation* is same as *ToLocation* and *time_end* specify the last reading time of the bag at this final location which is $time_{out}$ at *FromLocation*. The duration taken by a bag to go from *FromLocation* to *ToLocation* is stored in the *Duration* attribute. If *FromLocation* to *ToLocation* both are from same airport then the

value of *Duration* also represents the time spent by a bag at *FromLocation*.

We store *HandlerID* for each movement of a bag to track the responsible handler of the bag for that specific location. If it takes longer than expected to transfer the bag between the tracked location or the movement of the bag is not correct, then it will be possible to know which handler actually handled that section and this mishandling can be attributed to the quality of that handler. The status of a bag at each stay is stored in *StatusID* attribute. Storing both source and destination for each tracking record enables lot of opportunity for analysis of baggage flow from one location to another in an easy and efficient way. This technique also allows giving a status to each movement of a bag from one tag reader location to another. The *ResponsibleFlightID* column points to the *Flight* dimension to keep track which flight is responsible for the movement. It allows to find the baggage handling quality of different flights and airline. Additionally we store *FlightLegID* that points to *Dim_FlightRouteBridge* table. It helps to find how baggage handling quality affects for transit duration.

4.4 Cube Design

A multi-dimensional cube is built on top of the data warehouse described above. There can be various types of measures for answering different types of analytical questions on RFID baggage data. We give a few examples of analytical questions: a) *Average time taken by bags to move from sorter to gateway in Aalborg airport*, b) *Total number of bags handled by Copenhagen airport in the Christmas holiday*, c) *Total number of bag sent to wrong destination from Göteborg airport in weekends of January 2012*, d) *Maximum time taken by bags to go from check-in to sorter in Stockholm airport*. To answer these queries we used the following measures.

Duration, maximum duration, minimum duration, average duration. The *duration* column itself is used as measure and also enable other relevant measures like *maximum duration, minimum duration, total duration* taken by objects to go from one location to another. The *Duration* is also used for a computed measure *avg duration* i.e., the average duration taken by objects to move between locations. These measures are semi-additive and should be used with appropriate dimension to get relevant results. Because this is not meaningful to find the average, maximum, minimum duration without considering the *FromLocation* dimension. Moreover finding the sum of duration regardless of *Bag* dimension is also not meaningful. The use of maximum duration measure is shown in the following example MDX query in Q1. It returns a matrix showing max time taken by all bags to go from one tag reader location to another at AAL. In the query the parameter 2 for *descendants* function indicates two levels down from the airport level i.e., the tag reader level.

Listing 2.1: Q1

```
SELECT DESCENDANTS([FromLocation].[LocationHierarchy
   ].[Airport].&[AAL], 2) ON Axis(1),
DESCENDANTS([ToLocation].[LocationHierarchy].[Airport
   ].&[AAL], 2)  ON Axis(0)
FROM [cube_name]
WHERE [Measures].[Maximum Duration]
```

BagID distinct count. The number of distinct *BagIDs* in the fact table and grouping this on interesting dimensions give lot of interesting results. For example, the number of bags traveled from one tag reader location to another, and one country to another or region to region, number of bags mishandled at different airports, number of bags traveled in different routes, etc. For example the following MDX query, Q2 shows the number of bags mishandled at different airports and the result is ascending order based on number of bag(s) mishandled.

Listing 2.2: Q2

```
SELECT [Measures].[Bagid Distinct Count] ON COLUMNS,
ORDER(NonEmpty([FromLocation].[IATACode].members),[
   Measures].[Bagid Distinct Count], DESC) ON ROWS
FROM [cube_name]
WHERE DESCENDANTS([Dim Status].[Status Group].&
   Mishandled, 0)
```

Fact stay count. The number of stay records in the fact table can be used for many types of computed measures like percentage calculation, average calculation, etc. For example finding the average duration taken by bags to go from one tag reader location to another at Aalborg airport can be calculated by the following MDX query, Q3 where we use the measure Fact Stay Count.

Listing 2.3: Q3

```
WITH MEMBER[measures].[avgDuration]
AS '[Measures].[Duration]/[Measures].[FactStayCount]'
SELECT DESCENDANT([FromLocation].[LocationHierarchy].[
   Airport].&[AAL], 2) ON Axis(1),
DESCENDANTS([ToLocation].[LocationHierarchy].[Airport
   ].&[AAL], 2)  ON Axis(0)
FROM [cube_name]
WHERE [Measures].[avgDuration]
```

Incorporating Flight and Bag many-to-many relationship into the cube. Incorporating many-to-many relationship into a cube needs extra care. Because improper relationship between the dimensions and measures produce wrong aggregation results. As mentioned earlier we store both planned route and actual route of bags. The actual route taken by a bag is found from the tracking records stored in the fact table *Fact_stay*. On the other hand a long chain of relational tables are maintained between *Flight* and *Bag* table for planned route and flights. To get the aggregate results like the number of bags planned for traveling by particular flight or airline, the *Dim_Bag* table is used as fact table and a measure *Bag Count* is created on this fact. Here the dimensions to be used by the measure are *Dim_Bag, Flight* and *BagFlightRoute*. As *Dim_Bag* and *Flight* is related by the *Dim_FlightRouteBridge* table through *BagFlightRouteID*, an intermediate fact table is required to incorporate them in the cube. So the *Dim_FlightRouteBridge* table is considered as an intermediate fact table where this fact is related with the *Flight* table by *FlightID* and to the *Dim_Bag* table by *BagFlightRouteID*. Now the *Bag Count* measure of *Dim_Bag* has a many-to-many relationship in the cube through the intermediate fact table *Dim_FlightRouteBridge*. The implementation of this concept in MS Analysis Services can be seen in Appendix A.1. An example query on this relation can be *total number of bags planned to be carried by each airline* and this can be answered by the following MDX query Q4.

Listing 2.4: Q4

```
SELECT [Measures].[Dim Bag Count] ON COLUMNS,
       [Flight].[Airline].members ON ROWS
FROM [cube_name]
```

5 ETL Design

In this section, we present the Extract-Transform-Load (ETL) design for loading relevant data from source databases to the data warehouse. The design is shown in Figure 2.7.

First *Dim_Location_Airport* is loaded with the relevant airports as this table is independent of any other dimension. This table is also loaded with a default airport *"AllAirports"* and an airport *"INVALIDAirport"*. The default airport is used as a reference from the *Date* dimension to handle UTC date and time. On the other hand *INVALIDAirport* is referred when there are some records containing some invalid values or null values. The *Dim_Location_TagReader* is loaded with tag reader information from the source data. Here also we have a default tag reader *UnknownReader* which points to *INVALIDAirport*. The name of the tag reader is made self contained, e.g., check-in1 reader of

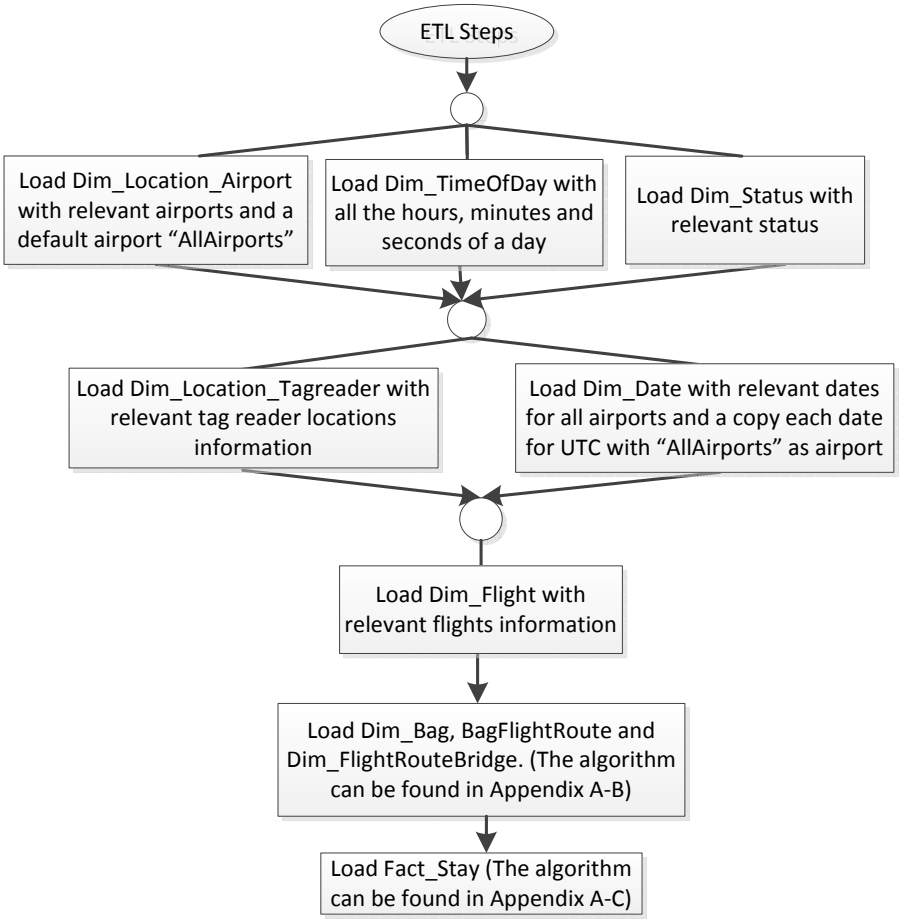


Fig. 2.7: Steps of ETL operation

Aalborg Airport of Denmark is stored as *'DK.AAL.Check-in'*.

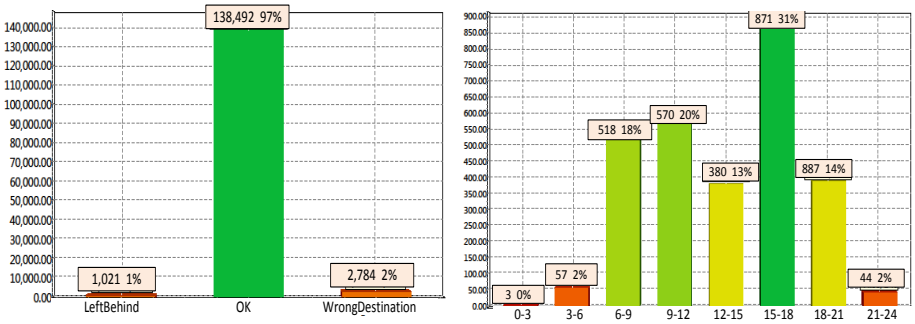
The *Date* dimension is preloaded with relevant years and the *TimeOfDay* dimension is preloaded with each second of a day. Both *Date* and *TimeOfDay* dimension is loaded for the airports where RFID readers are deployed and reading records are coming from these airports. There is a copy of Date for the default airport which is discussed earlier.

The *flight* table is loaded directly from the source data where date and time of departure and arrival is pointed to the *Date* and *TimeOfDay* dimension respectively. The departure airport and arrival airport points to the *Dim_Location_Airport* table.

The *Bag, BagFlightRoute* and *Dim_FlightRouteBridge* are loaded together at

the same time from the source data. The detailed algorithm can be seen in Appendix A.2.

The fact table *Fact_Stay* is loaded by the tracking records and relevant data from the source schema. Some preprocessing like creating some views in the source data can make the loading steps much easier and faster. We create two database view for this purpose. One view is created inside data warehouse schema with structure: *v_bag_flight*⟨*BagID, FlightId, sequenceIndex, FromAirportID, ToAirportID, ActualDepartureTimeUTC, FlightLegID*⟩ where the columns are taken by joining *Dim_Bag, Flight, Dim_FlightRouteBridge* tables of data warehouse schema. Another view is created inside source schema with structure:*v_track_airport*⟨*BagID, LocationID,* t_{in}, t_{out}, *AirportID*⟩, where the columns are taken from the location tracking records of Table 2.2 and *Airport* Table of source schema. The *v_track_airport* includes *AirportID* which helps to easily track, when actually a bag changed its airport as well as tracking the responsible flight of a bag for the tracked airport. The overall algorithm to load the fact table can be seen in Appendix A.3.

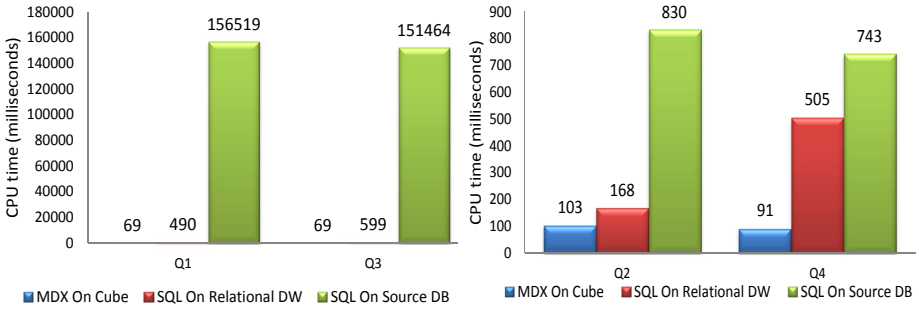


(a) Overall baggage status **(b)** WrongDest/DaySectionsIn3Hours

Fig. 2.8: Some analysis results

6 Experimental Results

We implemented the relational data warehouse in SQL Server 2008 R2 and the cube in SQL Server Analysis Services 2008 using MS Business Intelligence Development Studio 2008. The source data is stored in Oracle 11g. The ETL loads the source data from Oracle into the SQL Server DW and is implemented in C# with .Net framework 4.0. For advanced querying and visualization, TARGIT BI suite 2K11 is used. The experiments are conducted on a laptop with an Intel Core i7 2.7 GHz processor with 8 GB main memory. The operating system is Windows 7 64 bit. We use RFID-based airport baggage tracking data collected from 57 RFID readers deployed at 57 different



(a) Query Q1 and Q3 **(b)** Query Q2 and Q4

Fig. 2.9: Performance study

baggage handling locations at 7 airports in 7 cities in 2 countries. There were 149K bags with 4M filtered RFID readings. There was a huge data reduction after conversion of the RFID readings into location tracking records where it becomes 483K records. The readings were taken between 15-Dec-2011 and 17-Apr-2012 and the bags traveled on 36K flights.

Example Analysis Results We have generated many analysis results from the cube using through TARGIT BI suite, some are described here. The bar chart of Figure 2.8a shows *bagID distinct count* per *Status* where the status is one of *{OK, Left Behind, and WrongDestination}*. This shows that 97% of the bags are OK, 2% bags sent to the wrong destination, and 1% is left behind. Figure 2.8b shows the value of *bagID distinct count* grouped by the dimensions *{Status, and TimeOfDay}*. The status is drilled down to *Wrong Destination* and the *TimeOfDay* is at the *DaySectionsIn3Hours* level. This shows that among the 2% bags sent to wrong destination, 31% of the mishandlings happened between 15 PM to 18 PM, i.e., improvement efforts should focus here.

Performance We tested the above MDX queries on the MS Analysis Services cube and produced equivalent SQL queries for both of the source database and the relational data warehouse. To allow direct comparison, the source data was migrated from Oracle into SQL Server 2008. To observe MDX query performance, MDX Studio [7] was used. To observe SQL query performance, SQL Server 2008 Profiler was used and the queries is executed in SQL Server Management Studio. Each query is executed 5 times and for each execution, the query editor software is restarted and the cache is cleared. Finally, the average execution time is reported in Figure 2.9a and 2.9b for each query.

Query Q1 and Q3 is reported together in Figure 2.9a as both of them works with duration between tag reader locations of Aalborg airport. For the source data the value of duration, FromLocation and ToLocation is calculated using complex SQL queries and some intermediate views on the filtered

RFID readings. Due to the complexity some criteria e.g., the final location of an object is not considered (In DW FromLocation and ToLocation is same for the final location of an object). The results show that, for Q1, the CPU time on cube is 7 times faster compared to relational data warehouse and 2.3K times faster compared to source data. For the same query CPU time on relational data warehouse is 313 times faster compared to source data. Similarly for other queries, the results show that, the CPU time for all of the cases cube is significantly faster whereas the source database is significantly slower. The queries on relational data warehouse are also much faster compared to source database. The queries on the source database are very slow due to the huge data volume which is also not structured for analysis purposes. On the other hand, the proposed relational data warehouse is specially structured for analysis purposes and we also have reduced the data volume. Finally, the designed cube employs specialized MOLAP storage and has pre-computed aggregate results, which make the queries much faster. The differences between the execution times will become even larger for larger data sets.

7 Conclusion and Future Work

In this chapter we proposed a data warehouse solution for analysis of RFID-based airport baggage tracking data. We designed DW, including a number of complex dimensions and measures, in order to provide insight into the baggage tracking data and the reasons of baggage mishandling. We localized each date to a particular airport to capture special events related to that airport. We handled the many-to-many relationship between *Bag* and *Flight* dimension effectively both in the relational data warehouse and cube the design. The proposed data warehouse concepts can be used in similar types of indoor tracking application.

Future work will aim to solve the challenges encountered during this work as well as planned work. One important aspect is to scale the solutions to handle data from thousands of airport over long time periods, along with more precise capturing of the bag movement. This will require developing new pre-aggregation techniques. Another important aspect is to develop native support for *spatio-temporal sequences*, e.g., flight sequences, within the DW and BI tools, in order to get both more seamless querying and better performance.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

A Appendix

A.1 Implementation of Many-To-Many Relationship into Cube

Implementation of many-to-many relationship into cube in MS Analysis services is shown in Figure 2.10. The figure shows the dimension usage of SQL Server Business Intelligence Development Studio 2008 for the cube containing many-to-many relationship between *Bag* and *Flight*. For generating the figure, a Visual Studio add-in BIDS Helper [2] is used. From the figure we can see that *Dim_Bag* is used as both fact and dimension. Two measure groups on *Dim_Bag* and *Dim_FlightRouteBridge* are created. For *Dim_Bag* measure group the *Flight* dimension is used as many-to-many relationship where *Dim_FlightRouteBridge* is used as intermediate measure group. The *BagFlightRoute* is in regular relationship with the *Dim_Bag* measure group. Here the dimension column is *BagFlightRoute.BagFlightRouteID* and measure group column is *Dim_Bag.BagFlighRouteID*. Both *Flight* and *BagFlightRoute* dimension is in regular relationship with *Dim_FlightRouteBridge* measure group. For *BagFlightRoute* dimension the dimension column is *BagFlightRoute.BagFlightRouteID* and measure group column is *Dim_FlightRouteBridge.BagFlightRouteID*. On the other hand for *Flight* dimension the dimension column is *Flight.FlightID* and measure group colum is *Dim_FlightRouteBridge.FlightID*.

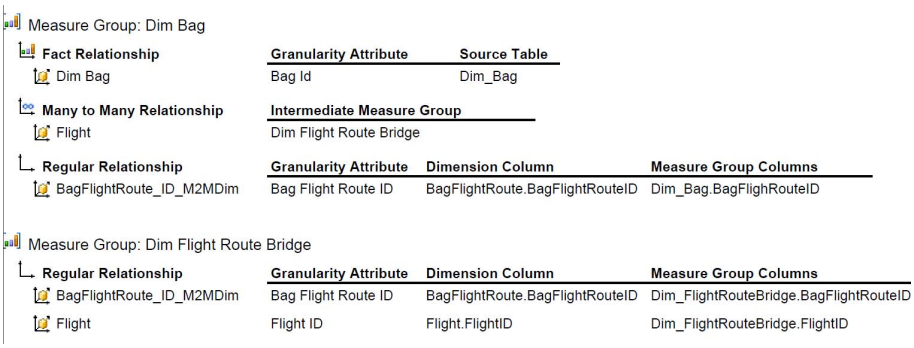


Fig. 2.10: Incorporating Flight and Bag relationship into cube

A.2 Algorithm: LoadBag_BagFlightRoute_FlightRouteBridge

The algorithm for loading *Dim_Bag, BagflightRoute and Dim_FlightRouteBridge* is shown in Algorithm 18 . In the algorithm at line 1, all the bags information is stored into variable*B* in the form $\langle BagID, LicensePlate, FlightDate, Priority \rangle$ as *BagID* wise ascending order. It helps to track easily which bags are already inserted into the data warehouse. If any error/failure occurs at the mid step during loading, we can quickly know which bags are already loaded by

checking only the last inserted *BagID*. Putting additional condition: *Where BagID>Last inserted BagID* in line 1 of the algorithm can avoid reloading of existing bags and decreased the loading time. For each bag in the source data the algorithm gets all the flight info and the sequence of flights the bag has planned to travel. To find out the existence of a *BagFlightRouteID* for the bag with exact same number of pairs of ⟨*FlightId, SequenceIndex*⟩ we use the concept of relational algebra for *exact relational division* [63] or *relational division without reminder*. A general relational division does not consider exact number of pair. For example consider the data of table *Bag-Flight Bridge* of Figure 2.6. Let us assume in *Bag-Flight-Route Bridge* table there be only one *BagFlightRouteID* exist which is 1: {⟨*F1, 1*⟩, ⟨*F1,2*⟩}. To find out the existence of *BagFlightRouteID* for bag *B3* in *Flight-Route Bridge* table, the relational algebra expression for general division operation can be:

$$\frac{\Pi_{BagFlightRouteID,FlightID,Sequence}\langle \textit{Flight-Route Bridge}\rangle}{\Pi_{FlightID,Sequence}\sigma_{BagID=B3}\langle \textit{Bag-Flight Bridge}\rangle}$$

For example the operation: {⟨*1, F1, 1*⟩, ⟨*1, F1,2*⟩}÷{⟨*F1, 1*⟩} results 1. It seems that trying to find out the existence of *BagFlightRouteID* for bag *B3* by the given general division operation returns 1 which is not correct. Because *BagFlightRouteID* 1 contains more flight sequences and we need to get a *BagFlightRouteID* containing exactly the set of value pair⟨*FlightID, Sequence*⟩ of bag *B3* i.e., { ⟨*F1, 1*⟩}.

To match the exact pairs it is important to count the number of pair(s) at divisor and divide it only with those dividend which contain the same number of pair(s) as the divisor. The concept of exact relational division is implemented by a complex SQL query shown in line 6 of Algorithm 18. Getting any row by executing this SQL indicates there exist a *BagFlightRouteID* with the given set of pairs (*U* of line 5 in Algorithm 18). If the SQL returns any *BagFlightRouteID* for this SQL, it inserts the bag with the returned *BagFlightRouteID* into the *Dim_Bag* table (line 8). In contrast with that, if the SQL does not return any *BagFlightRouteID* then a new record is inserted into the *BagFlightRoute* table with a new *BagFlightRouteID* (line 11). In line 13 it inserts the bag with this new *BagFlightRoute*.

A.3 Algorithm: LoadFactStay

The algorithm for loading the *Fact* table *Fact_Stay* is shown in Algorithm 25. As mentioned in the chapter we have used some views to make the steps easy. The structure of the views are: *v_bag_flight*⟨*BagID, FlightId, sequenceIndex, FromAirportID, ToAirportID, ActualDepartureTimeUTC, FlightLegID*⟩ and *v_track_airport*⟨*BagID, LocationID,* t_{in}, t_{out}, *AirportID*⟩. At the beginning of the

Algorithm 1: LoadBag_BagFlightRoute_FlightRouteBridge()

1 B := Set of all bags from the source data in the form ⟨*BagID, LicensePlate, FlightDate, Priority*⟩ in bagid wise ascending order;
2 BR := Set of routes of all the bags in the form ⟨ *Bagid, Flightid, Sequenceindex* ⟩ from the source data;
3 **for** *each record* $b_i \in B$ **do**
4 R := all the route info of bag b_i.*Bagid* from BR;
5 $U := \bigcup_{j=1}^{SizeOf(R)}$ ⟨ R_j.*flightid*, R_j.*Sequenceindex*⟩;
6 *result* := ExecuteQueryInDataWarehouse ⟨ *"WITH pair AS (SELECT U)*
7 *SELECT b.BagFlightRouteID as BagFlightRouteID*
8 *FROM Dim_FlightRouteBridge b LEFT JOIN pair p*
9 *ON (b.SequenceIndex = p.SequenceIndex AND b.FlightID = p.FlightID)*
10 *GROUP BY b.bagFlightRouteID*
11 *HAVING count(CASE WHEN p.flightid IS NULL THEN 1 END) = 0 AND Count(*) = (SELECT count(*) FROM pair)"* ⟩;
12 **if** result.RowCount $\neq 0$ **then**
13 Insert into *Dim_Bag*⟨ b_i.*Bagid*, b_i.*Licenseplate*, *result.BagFlightRouteID*, b_i.*Priority*⟩;
14 **else**
15 Generate FlightString and RouteString from BR and flight's source and destination information;
16 Add a new record to *BagFlightRoute* table with a new *BagFlightRouteID*. This can be done by using database sequence or auto generated identifier;
17 Add records to *Dim_FlightRouteBridge* table with all the values of U and the new *BagFlightRouteID*. For *TransferDuration* subtract actual arrival time of current flight sequence from the schedule departure time of next flight. For the final flight of a *BagFlightRouteID* put *TransferDuration* as 0 (Zero);
18 Insert into *Dim_Bag*⟨b_i.*Bagid*, b_i.*Licenseplate*, *The new BagFlightRouteID*, b_i.*Priority*⟩;

algorithm the tracking records, distinct *BagID* and flight info is loaded into variables (lines 1-3).

For each bag the algorithm retrieves and determine necessary fields for each tracking record and insert it into *Fact_Stay* table (lines 4-27). For each bag the algorithm filters the records of the particular bag and keep them into variables (lines 5-6). As for the final tracking record of a bag the *FromLocation* and *ToLocation* is same in the *Fact_Stay* table, we separate that tracking record into a variable R_{end} (line 7). This record is handled separately for each bag (lines 22-27). For each tracking record of a bag except R_{end} (line 8), the algorithm determine the *FromLocation* and *ToLocation* (line 10), $time_{start}$ and

Algorithm 2: LoadFactStay()

```
1  TR := Set of all tracking records from v_track_airport in BagID and t_in wise
   ascending order;
2  B := Set of distinct BagID in TR on BagID wise ascending order;
3  BF := Set of all flight records from v_bag_flight in BagID and SequenceIndex wise
   ascending order;
4  for each BagID b_i ∈ B do
5      F := All records for bag b_i from BF;
6      R := All tracking records for bag b_i from TR;
7      R_end := Last record of R; seq := 1;
8      for each record r_i ∈ R \ R_end do
9          fromLoc := r_i.LocationID; toLoc:= r_{i+1}.LocationID;
10         fromAirport := r_i.AirportID; toAirport:= r_{i+1}.AirportID;
11         time_start:=r_i.t_in;time_end:=r_{i+1}.t_in;
12         Duration := time_end - time_start; Status := "OK";
13         if fromAirportID = toAirportID then
14             Find the status of the movement and the responsible flight with the
               help of F.;
15         else
16             if toAirportID≠ Airport of planned destination. Check this from F with
               the help of seq then
17                 Status := "WrongDestinaion";
18                 Find ResponsibleFlightID from F with the help of seq;
19                 seq++;
20         Insert a new record into the Fact table with b_i, fromLoc, toLoc, StatusID
           for Status, Duration, ResponsibleFlight, DateID and TimeID of time_start,
           time_end for both UTC, fromAirportID and toAirportID;
21     Now for the last tracking record R_end, fromLoc := R_end.LocationID; toLoc :=
       fromLoc;
22     time_start := R_end.t_in; time_end := R_end.t_out;
23     Duration := time_end - time_start; Status := "OK";
24     Find the status and other attributes based on the above logic;
25     Insert a new record into the Fact table with b_i, fromLoc, toLoc, StatusID for
       Status, Duration, ResponsibleFlight, DateID and TimeID of time_start, time_end
       for both UTC, fromAirportID and toAirportID;
```

$time_{end}$ (line 11), check whither the bag changes the airport or not (lines 13 and 16) and based on the information it determines the *Status* of that particular tracking record (lines 14 and 17). The variable *seq* is used to track how many airport is changed by the bag which helps to retrieve flight info for that particular sequence and also to determine the *Status*. For R_{end} the operations are very similar to the other records except the value of *ToLocation*

(line 21) and $time_{end}$ (line 22). After retrieving and determining the values of necessary fields a record is inserted into *Fact_Stay* (lines 20 and 25).

Chapter 3

Mining Risk Factors in RFID Baggage Tracking Data

The paper has been published in the
Proceedings of the IEEE 16th International Conference on Mobile Data Management(IEEE MDM 2015), Pittsburg, Pennsylvania, USA, pp. 235–242, 2015. The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1109/MDM.2015.31

Abstract

Airport baggage management is a significant part of the aviation industry. However, for several reasons every year a vast number of bags are mishandled (e.g., left behind, send to wrong flights, gets lost, etc.,) which costs a lot of money to the aviation industry as well as creates inconvenience and frustration to the passengers. To remedy these problems we propose a detailed methodology for mining risk factors from Radio Frequency Identification (RFID) baggage tracking data. The factors should identify potential issues in the baggage management. However, the baggage tracking data are low level and not directly accessible for finding such factors. Moreover, baggage tracking data are highly imbalanced, for example, our experimental data, which is a large real-world data set from the Scandinavian countries, contains only 0.8% mishandled bags. This imbalance presents difficulties to most data mining techniques. The chapter presents detailed steps for pre-processing the unprocessed raw tracking data for higher-level analysis and handling the imbalance problem. We fragment the

data set based on a number of relevant factors and find the best classifier for each of them. The chapter reports on a comprehensive experimental study with real RFID baggage tracking data and it shows that the proposed methodology results in a strong classifier, and can find interesting concrete patterns and reveal useful insights of the data.

1 Introduction

Aviation industry suffers from enormous loss due to baggage mishandling. A recent report [10] shows that in 2013, 3.13 billion passengers traveled by airlines and among them around 21 M passengers and 21.8 M bags were affected by baggage mishandling that costs 2.09 billion USD to the airline industry. It also creates frustration to the passengers during their vacation or business trips. Common baggage mishandlings are: left behind at the origin airport, missed connecting flight, bag loss, wrong bag destination, etc. A bag has to follow several steps while traveling from the origin airport to the final destination. The steps include check-in, screening, sorting, loading, transfer at the transit airport, arrival, etc. Mismanagement at any of these stages can be the reason for the bag being mishandled. The use of Radio Frequency Identification (RFID) in the baggage management system enables to track a bag while passing through different stages within an airport as well as across the airports. The massive baggage tracking can be very useful for analyzing and finding interesting patterns. Combining the tracking data with other dimensions like route information, flights and punctuality, day hours, week day, transit duration, etc., can reveal risk factors that are responsible for baggage mishandling.

Data mining is the process of extracting interesting patterns and knowledge from large data sets, and the acquired knowledge can be used for predicting unknown labels of new instances [42]. For example, from the baggage tracking data, we may be interested in knowing, *what is the probability that a bag will be mishandled if it has 35 minutes transit time at Copenhagen airport on Sunday morning*? However, before performing any data mining the data has to be well prepared such that the gained knowledge is useful. In the baggage tracking scenario, the generated huge volume tracking data are very low level and not directly suitable for further analysis. Therefore, relevant and important features need to be extracted from the unprocessed raw tracking data. Furthermore, the percentage of mishandled bags is very low (e.g., only 0.8% in our experimental data set) as compared to the percentage of correctly handled bags. It makes the data set highly imbalanced. This imbalance problem in the data set makes the mining process biased towards predicting that any bag will be handled correctly by default and makes it difficult to learn rules related to incorrectly handled bags. Thus, we need to take special care of this issue to get the mining techniques to work properly and to get patterns of higher quality.

In this chapter, we propose a step by step methodology for performing data mining tasks to find interesting patterns and risk factors that are highly correlated with baggage mishandling. We present the essential steps for extracting a set of high-level features called *FlightLeg Records* for mining from

the unprocessed raw RFID baggage tracking data. We have applied various classification techniques on the data set and deal with the imbalance problem by applying several different re-balancing techniques for finding the best predictive model. We also fragment the data set based on some important factors and learn specialized classification models for each fragment. We have conducted a comprehensive experimental study with a large amount of real-world RFID baggage tracking data from a major industry initiative called the BagTrack project (www.daisy.aau.dk/bagtrack). The data set has been collected from several airports in the Scandinavian countries. The experiment shows that fragmenting the data set helps to achieve better models. We also analyze and report some interesting patterns and risk factors that are discovered from the data set. The proposed methodology and techniques can help the aviation industry for examining baggage management problems and ultimately improving the baggage handling quality.

The remainder of the chapter is organized as follows. Section 2 presents the preliminaries including the problem statement. Section 3 discusses the steps of the solution. Section 4 reports the experimental results. Section 5 reviews related work. Section 6 concludes and points to future work.

2 Preliminaries

RFID-Based Baggage Handling In airport baggage management a bag has to go through different steps to go from origin to final destination. Fig. 3.1 shows some airports and stages involved in baggage management at multiple airports. Suppose that Nadia needs to travel from Aalborg Airport (AAL) to Arlanda Airport (ARN) via Copenhagen Airport (CPH). First, Nadia has to check-in and handover her bag to the check-in desk staff. Then the staff puts the bag on the conveyor belt for the automatic baggage sortation system. After passing all the steps inside AAL, the bag is loaded into the aircraft using belt loader for the targeted flight. As the bag has to be transferred to ARN, upon arrival at CPH it is shifted to the transfer system. After all the required stages at CPH, the bag is loaded to the aircraft for its next flight to ARN. After arriving at ARN, the bag is shifted to arrival belt and finally Nadia collects the bag from the arrival belt. During this journey, the bag has to go through up to 11 stages and there can be many baggage handlers handling the bag at the different stages.

Fig. 3.2 shows an example of RFID reader deployment at different locations of a baggage management system. An RFID reader is deployed in a fixed location and the position of the reader is recorded in the database. For example, *reader1* in Fig. 3.2 corresponds to *check-in1*, *reader6* corresponds to *Gateway-1* etc. The circles represent the RFID readers and their activation ranges.

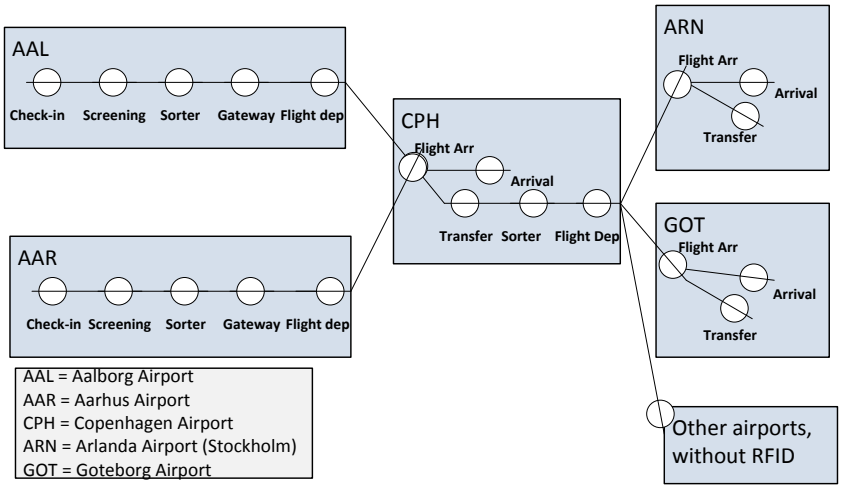


Fig. 3.1: Baggage flow in multiple airports

At check-in, an RFID tag is attached to the bag. The tag contains a small built in memory that stores bag information including the bag identifier, flights, route legs, date of departure, etc. While passing different stages, whenever a bag enters into a reader's activation range, it is continuously detected by the reader with a sampling rate which generates raw reading records of the form: $\langle BagID, Location, Time, \{info\}\rangle$, meaning that a reader at location *Location* detects a bag with ID *BagID* at timestamp *Time* and the tag stores the information *info*. Considering only location and time related information, some examples of raw reading records are shown in Fig. 3.3a. In the table, *RID* represents the reading identifier. As seen a bag can have several readings at the same location and on the basis of a single record, it is also not directly possible to compute how long an object spent in a particular location. To overcome these problems and prepare the data for further analysis we convert the raw reading records into stay records [15].

Stay Records A stay record is of the form: $StayRecord\langle BagID, FromLocation, ToLocation, t_{start}, t_{end}, Duration, \{StayInfo\}\rangle$ which represents that a bag with *BagID* first appeared at *FromLocation* at time t_{start} and then first appeared at the next location *ToLocation* at time t_{end}. It took *Duration* time to go from the reader at *FromLocation* to the reader at *ToLocation*. The *{StayInfo}* represents a set of other dimensional information related to the bag and to the transition (e.g., bag status, next flight schedule, origin and destination airports, and other flight-related information). For the final location of a bag, a special stay

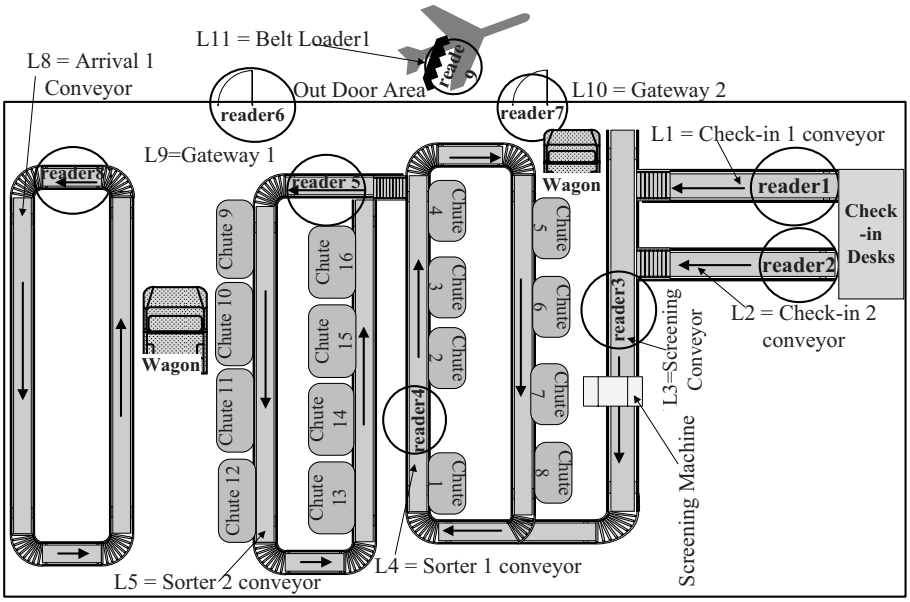


Fig. 3.2: RFID reader deployment in airport baggage tracking [15]

record is stored where the *FromLocation* and the *ToLocation* are same and the t_{start} and t_{end} represent the first and last times the bag appeared at that location. The stay records compress the huge data volume of raw readings and also enable to find abnormally long time spans between locations that may lead to baggage mishandling. Fig. 3.3b shows the stay records for the raw records of Fig. 3.3a. In the table, *RecID* represents the stay record identifier. In Fig. 3.3b, *Rec1* represents that bag *B1* had a transition from *AAL.Checkin-1* to *AAL.Screening* and it took 3 time units for the transition. The bag first appeared at *Checkin-1* at time 1 and then appeared at *Screening* at time 4. The stay records mainly introduce a very important feature which is the duration information. Nevertheless, it is still at a lower level for further higher level analysis. We are more interested in a higher level analysis like the baggage management performance at airport level, weekday, hour of day, and other relevant factors. We take the duration feature including some other dimensions from the stay records and create a table called *FlightLeg Records* which is described below.

FlightLeg Records The attributes and their descriptions of the *FlightLegRecords* table are shown in Table 3.1. For each flight of a bag we have one instance in the *FlightLegRecords* table. It captures some important features extracted from the *Stay Records* like {*IsTransit, DurationBeforeFlight, IsLongerStayFound, TotalBagInThatHour, Status*}. The way of calculating the value of *DurationBeforeFlight* varies based on whether the record belongs to transit or

(a) Raw Reading Table

RID	BagID	Location	Time
R1	B1	AAL.Chkin1	1
R2	B1	AAL.Screen	4
R3	B1	AAL.Screen	5
R4	B1	AAL.sorter1	8
R5	B1	AAL.sorter1	9
R6	B1	AAL.Gate2	15
R7	B1	AAL.BltLd1	21
R8	B1	CPH.Trans1	70
R9	B1	CPH.Tran1	72
R10	B1	CPH.Sorter2	80
R11	B1	CPH.Sorter2	90
R12	B1	CPH.Sorter2	100

(b) Stay Records

Rec ID	Bag ID	FromLoc	ToLoc	t_{start}	t_{end}	Dur.
Rec1	B1	AAL.Chkin1	AAL.Screen	1	4	3
Rec2	B1	AAL.Screen	AAL.sorter1	4	8	4
Rec3	B1	AAL.sorter1	AAL.Gate2	8	15	7
Rec4	B1	AAL.Gate2	AAL.BeltLd1	15	21	6
Rec5	B1	AAL.BltLd1	CPH.Tran1	21	70	49
Rec6	B1	CPH.Trans1	CPH.Sorter2	70	80	10
Rec7	B1	CPH.Sorter2	CPH.Sorter2	80	100	20

(c) FlightLeg Records

Bag id	From Airpt.	To Airpt.	Is Tran.	Weekday	Flight Time	DurBef. Flight	IsLongStayFound	Delay InArr.	TotBag ThatHr	Status
B1	AAL	CPH	0	Monday	9-10	25	0	NULL	86	OK
B1	CPH	ARN	1	Monday	10-11	30	1	-5	70	Mishan.

Fig. 3.3: Example of getting stay records from raw records and example of FlightLeg records

not (see description of *IsTransit* column). For a non-transit record, it is calculated from the first reading time of the bag at check-in and the actual departure time of the flight. Conversely, for a transit record, it is calculated from the actual flight arrival time at the *FromAirport* and actual departure time of the next flight to the *ToAirport*. The *DurationBeforeFlight* attribute is useful to see the effect on baggage mishandling due to the operating duration before departure. The value of *IsLongerStayFound* is determined by comparing the movement of baggage between readers at *FromAirport*. For each distinct transition between a pair of locations in the *Stay Records*, the bags that followed the top 5% longest durations are considered as *longer than expected*. The value of *DelayInArrival* is calculated from the actual and scheduled arrival times of the flight in the transit airport (i.e., *FromAirport* is a transit airport). For the non-transit records *DelayInArrival* is *NULL*. The status of the bag indicates whether the bag was mishandled or not in the *FromAirport*. The status of a bag is extracted from the reading records of the bag at the readers at *FromAirport*, flight timing, route information, etc. If a bag has any reading in the *FromAirport* after the corresponding flight departure time, then the bag is considered as left behind. Conversely, if a bag has any reading from an airport which is not in its planned route, then it is considered as wrong destination. Fig. 3.3c shows an example of the content of *FlightLegRecords*. We use the *FlightLegRecords* table for our further analysis.

Problem Statement Our primary goal is to find interesting patterns and identify risk factors that are correlated to baggage mishandling and ideally indicate appropriate corrective actions. We want to find bags with higher probability of being mishandled.

Table 3.1: Description of the attributes of *FlightLeg records*

No.	Name (Data Type)	Description
1	*FromAirport (Symbolic)*	Departure airport of the corresponding flight
2	*ToAirport (Symbolic)*	Destination airport of the corresponding flight
3	*IsTransit (Boolean)*	A Boolean value representing whether it is a transit bag or not for the corresponding flight
4	*Weekday (Symbolic)*	Weekday of the corresponding flight
5	*FlightTimeHour (Symbolic)*	Departure hour of the corresponding flight
6	*DurationBeforeFlight (Integer)*	Available time (in minutes) for the bag to catch the flight
7	*IsLongerStayFound (Boolean)*	If any stay duration between readers at the *FromAirport* is longer than expected then it is *true*, otherwise it is *false*
8	*DelayInArrival (Integer)*	Delay in arrival (in minutes) of the arrival flight for the transit bag
9	*TotalBagInThatHour (Integer)*	Total number of bags read during the departure hour of the flight at *FromAirport*
10	*Status (Symbolic)*	Status of the bag i.e., 'OK' or 'Mishandled'

Definition 1 (Risk Score). Given a set of *FlightLeg records* F and an instance $f \in F$, the risk score $r \in \mathbb{R}$ is the probability estimate (PE) score for the instance f being *Mishandled*.

Definition 2 (Rank). Given a set of *FlightLeg records* F with assigned risk scores, the rank of a record $f \in F$ is the position of f in the list of F sorted by risk in descending order, i.e., the record with the highest risk score is ranked first.

We are interested in finding the best predictive model that can produce correct risk scores and the most accurate ranking of our data set.

3 Solution

For producing a risk score as well as a ranking we take the help of a classification algorithm. In order to find a risk score, the system has to learn

from a set of training records and assign a risk score to each test example. We use the *Status* attribute of the *FlightLegRecords* as the class column. However, in our data set around only ***0.8%*** of the records belongs to *'Mishandled (MH)'* and the remaining ***99.2%*** of the records belongs to *'OK'*. With such an imbalanced distribution between the classes, the learning process gets biased towards predicting *OK* for almost all the instances by default and frequently misclassifies the *MH* instances. This is because a classifier tries to make the classification rules more general and considers the *MH* records as noise and discards the *MH* records. As a result, this *imbalance problem* should be handled wisely to overcome poor quality results otherwise produced by the classifier. It is also essential to choose an appropriate classification algorithm which will provide a good quality result for the given data set. To achieve the intended quality results from the raw baggage tracking data, we follow some essential steps. The steps of our solution are given in Fig. 3.4 and discussed in the following.

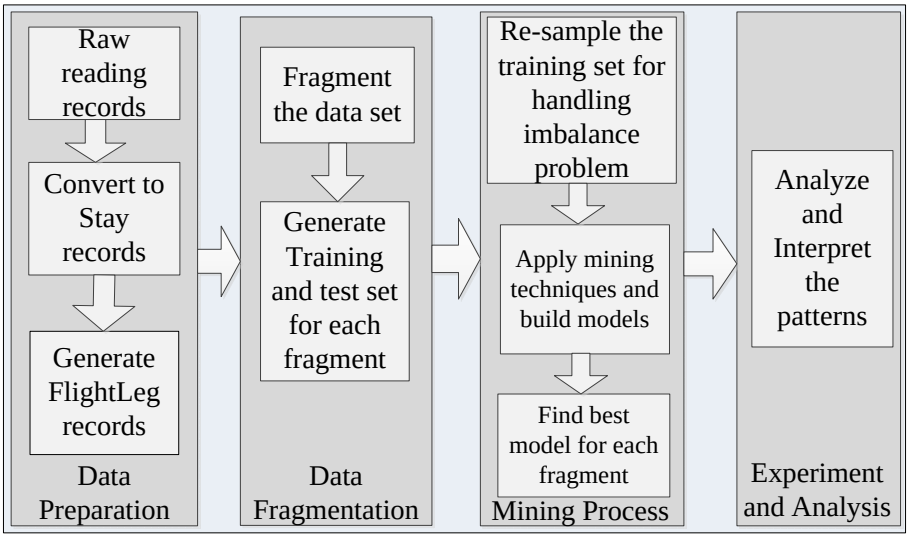


Fig. 3.4: Outline of the steps

3.1 Data Preparation

Before applying any data mining technique and finding patterns, we need to pre-process the source data and select and construct relevant features. In this step, the raw baggage tracking records are converted into *Stay records* and then into the *FlightLegRecords*. The structure of the tables and steps of preparing such tables were already described in section 2.

3.2 Data Fragmentation

Fragments The baggage management problem varies based on different important factors like whether the bag is in the transit airport or not, the duration of the transit, etc. Based on some important factors we have divided the data set into 5 fragments and applied data mining algorithms on each of the fragments for finding patterns specific to each fragment. Moreover, as the data set is imbalanced, the fragmentation allows examining the imbalance problems for specialized cases. Fig. 3.5 shows the different fragments of our data set, and the numbers inside the square bracket show the number of records and mishandling rate for the corresponding fragment in our experimental data. The combined records (CR) contain all the records of *FlightLegRecords* table. *CR* is divided into *transit records (TR)* and *non-transit records (NTR)*. To see the effect of transit duration on the baggage mishandling rate, we have drawn Fig. 3.6. It shows that almost all the bags (80-100 %) are mishandled when the transit duration is $\leq$31 (minutes). Based on transit duration, we have divided the *TR* into two different fragments. Transit records containing *DurationBeforeFlight*$\leq$*31* belong to fragment *Shorter Transit Records (STR)* and other transit records belong to fragment *Longer Transit Records (LTR)*. Both of these fragments help to analyze the shorter and longer transit baggage separately.

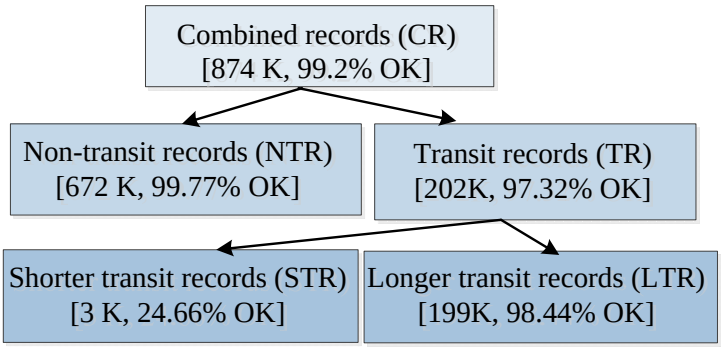


Fig. 3.5: Fragments of the data set

Training and test set Before applying data mining techniques, we have to prepare the training and test data sets. For each of the discussed fragments, we have one partition for training or learning (P1), and another partition for testing (P2). Strategies for obtaining the training and test sets are explained further in the experimental evaluation section.

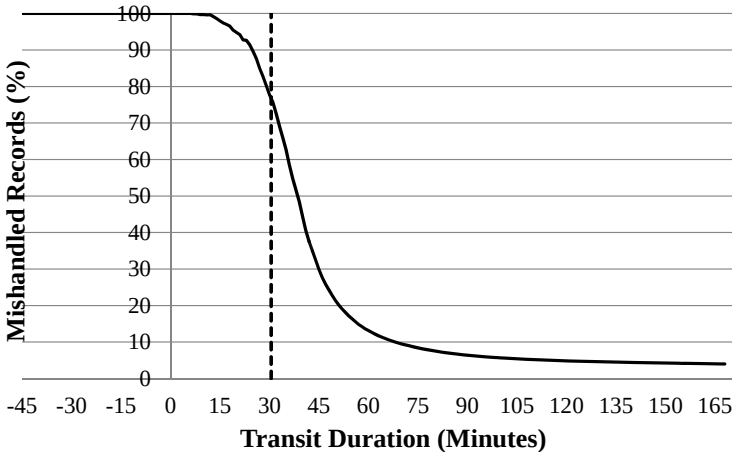


Fig. 3.6: Mishandling rate with change in transit duration

3.3 Mining Process

Handling imbalanced data by re-sampling As discussed earlier, that the data set is highly imbalanced and learning directly from the data set will produce very poor quality patterns. To remedy these imbalance problems, we use 2 different kinds of sampling for the training data set (P1):
***Undersampling technique* (US)**: in this technique a subset of *P1* is created by randomly deleting *OK* records until we reach equal number of records with class *OK* and class *MH*.
***Oversampling technique* (OS)**: in this technique a superset of *P1* is created by copying some instances or generating new instances of *MH* records until we obtain an equal number of records for class *OK* and *MH*. We use Synthetic Minority Over-sampling Technique (SMOTE) [28] for getting *OS* data.

Mining Techniques We apply *Decision Tree (DT), Naive Bayes classifier (NB), KNN classifier (KNN), Linear regression (LIR), Logistics regression (LOR)*, and *Support vector machine (SVM)* on the training set *P1* of the combined records *CR* with the sampling strategies discussed above. We also do the same directly to P1 without re-sampling *(WS)*. Then we use different types of measures (discussed in the next paragraph) for finding the classification and sampling techniques that provide the best model for our data set. Then the chosen techniques are used for generating models for the remaining fragments. Note that we have deliberately chosen not to consider association rules mining technique to find out rules based on confidence and support scores, since this is an unsupervised technique, while our problem is supervised. We are only interested in modeling our target variable w.r.t. the other variables. In some sense, decision tree induction can be considered as a form of rule induction; every path from the root to a leaf actually represents an

association rule.

Finding the best technique In general, a confusion matrix as shown in Table 3.2 is used for assessing the performance of a classifier. The confusion matrix shows how many test records are correctly and incorrectly classified for both positive and negative classes.

Table 3.2: Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	*True Positive (TP)*	False Negative (FN)
Actual Negative	*False Positive (FP)*	True Negative (TN)

Typically the performance of a classifier is evaluated by its predictive accuracy defined by, *Accuracy = (TP+TN)/(TP+FP+TN+FN)*. However, for an imbalanced data set the predictive accuracy is not an appropriate measure. For example, in our case an accuracy of 99% does not make sense, as it may misclassify all the examples as *OK (negative)* regardless of whether a record belongs to *Mishandled (positive)* or not. Here actually the classifier is 99% accurate in predicting the negative instances and 0% accurate in predicting the positive instances.

In an information retrieval system, *precision* is the measure which represents the relevance of the retrieved results, whereas *recall* represents the coverage of the relevant instances in the result. Precision is calculated as, *Precision = TP/(TP+FP)*, and recall (also known as TP rate) is calculated as, *Recall = TP/(TP + FN)*. A precision-recall (PR) curve is a good way to visualize the precision for a given recall. The points of a PR curve are calculated based on the generated scores of the classifier. The scores are sorted in descending order and each score is considered as a threshold for calculating the value of the precision and recall to draw a point in the PR curve.

The receiver operating characteristic (ROC) curve [68] is a well-known visualization of the performance of a ranker. The X-axis of an ROC curve represents false positive rate (FP rate = FP/(FP + TN)) and the Y-axis represents true positive rate (TP rate). So, it shows the trade-off between the TP rate and FP rate. The *Area Under the ROC Curve (AUC)* is a popular measure for evaluating the quality of ranking produced by a classifier [82]. The maximum value of an *AUC* can be 1 and it means that all the positive examples are placed in the top in the ranking. A classifier with AUC = 0.5 represents a random classifier that randomly guesses the classes.

In our cases, we consider two classes *'OK'* and *'Mishandled'* for classification and consider *'Mishandled'* as the positive class. In our scenario, misclassifying a *Mishandled* bag as *OK* (false negative) is more severe than misclassifying an OK bag as Mishandled. As such we are specifically interested in algorithms with a high recall on the *Mishandled* bags rather than in merely

optimizing the accuracy of the classifier. In our case, we use the AUC as the main measure for choosing the model that provides the best ranking. We use precision-recall curves for finding which threshold provides higher precision for a good amount of recall.

4 Experimental Evaluation

We use KNIME V2.9.2 [3] for modeling our experimental work flows, applying data mining algorithms, producing and visualizing the results obtained from our data set. For preparing the data set from the source data we use different kinds of SQL queries and C# programs.

We use real RFID-based baggage tracking data, collected from 13 different airports with a total of 124 RFID readers deployed. There are 196 M raw reading records for 1.4 M bags collected for the period from *January 1, 2012* until *December 2, 2013*. In the original data set there are a lot of incomplete and erroneous records, e.g., missing flight and route information, unusual reading time, reading from unknown readers, etc. It creates problems while extracting different information about a bag like transit information, status at different stages, delay in departure and arrival, flight time hour, etc. As a relatively clean data set is an essential part for data mining, the problematic bags with the above mentioned incomplete information are filtered out during conversion into *stay records*, leaving us with 728K bags with 2.68 M *stay records*. Among these bags, some bags have stay records only in the arrival airports, which do not create any instances in the *FlightLegRecords* table. Finally, after converting *stay records* into *FlightLeg records* we have 671,712 bags with 874,347 *flight leg records* for mining. Among them only 0.8% of the records belongs to the class *Mishandled* (MH), the remaining 99.2 % belongs to the class *OK*. For each fragment, the total number of records and percentages of *OK* are shown in Fig. 3.5. Among the fragments, only *STR* contains a higher number of *MH* records than *OK* records. In the rest of this section, we will show how different classification algorithms (discussed in Section 3.3) perform on the combined records with different kinds of re-sampling techniques. Then we will identify the best classification and sampling technique and discuss the obtained patterns and analysis results from the data.

For the Decision Tree Induction, the C4.5 algorithm is used with the *Gini index* quality measure and the *MDL* pruning method. To reduce the number of branches, the minimum number of records per node is set to 100. For the KNN classifier, we tried with K=5 and K=7. As in both of the cases the results were similar, we finally report for K=7. Before applying KNN, linear regression, logistics regression, and SVM the structure of the input data table is changed as these algorithms do not work with categorical attributes. In these cases, we convert each value of a categorical attribute into a separate

column and put Boolean 0 or 1 accordingly. An example of such conversion for Fig. 3.3c is shown in Table 3.3. For the linear regression and the logistics regression, the attributes with continuous values are normalized into the [0;1] interval. In our data set all the *FromAirport*s are within the *Schengen territory* [8] and a person traveling within this territory does not require any special passport control. Unlike *FromAirport*s we have too many values in the *ToAirport* column which creates many branches in the decision tree and for other classification algorithms this column become useless. To make the *ToAirport* column useful and make the learned pattern interesting we categorized the *ToAirport*s into three types: *Domestic, Schengen, and Others*.

Table 3.3: Converting string values into columns of Fig. 3.3c

AAL	CPH	IsTransit	Monday	9-10	10-11	....
1	0	0	1	1	0	...
0	1	1	1	0	1	...

In our experiments, we split the available data into a training and a test set based on the date of the record. All records before *15- May-2013* were included in the training set *(P1)* [Total: 615K, OK: 99.2%] and all records from that date or later were added to the test set *(P2)* [Total: 259K, OK: 99.18%]. The reason we did not rely on a standard cross-validation approach is because there exist dependencies between the bags. Bags that were on the same plane are more likely to have similar properties, as well as a similar class label. Therefore spreading bags of the same flight over both the training and test set may cause a biased estimation of the performance due to overfitting. By dividing the data based on date, we can guarantee that the training set and the test set are independent, and we get an unbiased estimate of the performance of the mined models.

Table 3.4: The table below lists the AUCs with the different types of classification algorithms and re-balancing/re-sampling techniques for the combined records (CR)

Re-sampling	DT	NB	KNN	LOR	LIR
WS	0.88	0.83	0.71	0.82	0.78
US	0.87	0.83	0.79	0.85	0.79
OS	0.81	0.83	0.78	0.83	0.74

Overall from Table 3.4 in all the cases we can see that the AUCs are better than a random classifier predicting by default class *OK* for every bag. It shows that the re-balancing technique (i.e., WS, US, and OS) has a high impact on the AUCs of some classifiers, whereas it has almost no effect for the *Naive Bayes classifier*. It shows that the decision tree produces the highest AUC compared to all the other classifiers. The AUCs produced by the over-sampling (OS) technique shows that it is not helping to produce a better

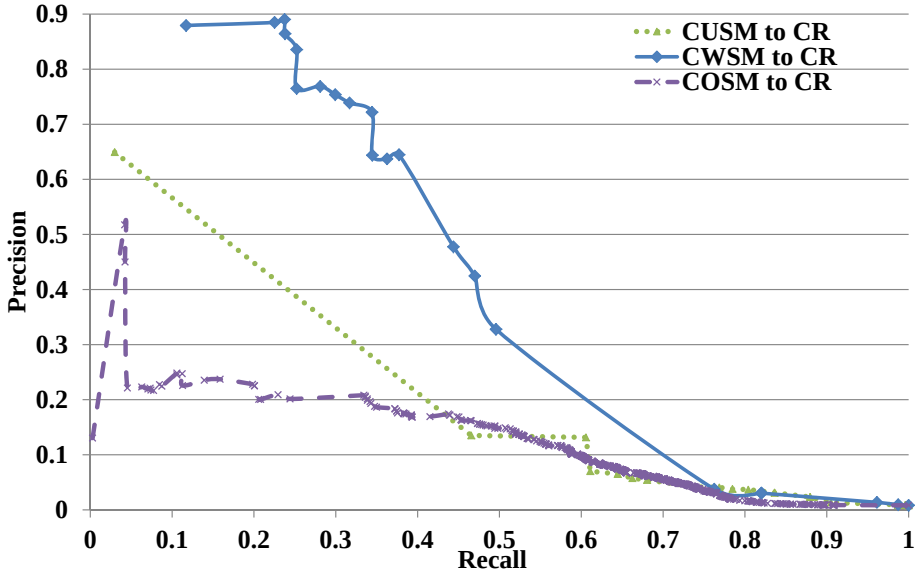


Fig. 3.7: Precision-recall curves for CWSM, CUSM, and COSM tested on the test set of the combined records (CR)

ranking in our data set. The performance of over-sampling is unpredictable given that without knowing the precise process that generated the data, it is hard to generate good synthetic examples. During our experiments, we found that the SVM learning process did not produce any results within a reasonable time (several days). So, we do not report any AUC with SVM. As the decision tree produces the highest AUC (i.e., the best ranking), we consider it as the best classifier for our scenario. For our further experiments, we will only use the decision tree *without re-sampling (WS)* and with *US* for producing other models. We call the model generated by the decision tree without re-sampling (*WS*) the Combined Without-sampling Model (CWSM), with *US* the Combined Under-sampling Model (CUSM), and with *OS* the Combined Over-sampling Model (COSM). Fig. 3.7 shows the precision-recall (PR) curves for *CWSM*, *CUSM*, and *COSM* when applied to the *CR*. It shows that *CWSM* produces the best PR curve that always gives a higher precision for the different values of recall compared to the other two models. It shows that for 50% recall, we can get 34% precision. It means that the ranking produced by the decision tree contains 35% of the actual *MH* records among the top 50% predicted *MH* records.

After finding the best classification algorithm and short listing the re-balancing techniques we conduct further experiments on the different fragments. We learn decision tree models for *NTR*, *TR*, *STR*, and *LTR* without re-balancing (*WS*) and respectively they are called:

Table 3.5: AUCs for models built from different fragments and testing on relevant fragments

Sampling	Model	CR	NTR	TR	STR	LTR
WS	**CWSM**	**0.88**	**0.74**	0.79	**0.79**	0.66
	NTWSM	-	**0.74**	-	-	-
	TWSM	-	-	0.67	0.72	0.5
	STWSM	-	-	-	0.73	-
	LTWSM	-	-	-	-	0.61
US	**CUSM**	0.87	0.67	0.82	0.54	0.77
	NTUSM	-	0.73	-	-	-
	TUSM	-	-	**0.85**	0.5	**0.78**
	STUSM	-	-	-	0.77	-
	LTUSM	-	-	-	-	**0.78**

- *Non-transit Without-sampling Model (NTWSM)*
- *Transit Without-sampling Model (TWSM)*
- *Shorter Transit Without-sampling Model (STWSM)*
- *Longer Transit Without-sampling Model (LTWSM)*

We also learn decision tree models for the fragments with *US* and respectively they are called:

- *Non-transit Under-sampling Model (NTUSM)*
- *Transit Under-sampling Model (TUSM)*
- *Shorter Transit Under-sampling Model (STUSM)*
- *Longer Transit Under-sampling Model (LTUSM)*

For all the cases, the training and test sets are taken by filtering the data from *P1* and *P2* of the *CR*. Then we apply the models *CWSM* and *CUSM* to the test sets of all these fragments. We also apply all the other models to the relevant fragments for finding the best models for each of the fragments. Table 3.5 shows the AUCs for the models and cross checking with the different fragments. It shows that the individual models give a reasonable AUC with their own fragment. The AUCs with the *TR* shows that both fragmenting and re-balancing helps to achieve better models for the transit cases. For the *NTR*, models without re-balancing (i.e, *CWSM* and *NTWSM*) produce better ranking. Fig. 3.8 shows the PR curves of different models when applied to the *NTR*. It shows that both *CWSM* and *NTWSM* gives very similar precision for different values of recall. So, from the AUCs and PR curves we can conclude that both *CWSM* and *NTWSM* can produce better ranking and patterns for the *NTR* compared to the other models. Table 3.5 shows that *CWSM* produces the highest AUC for the *STR*, and the next closer AUC is produced by

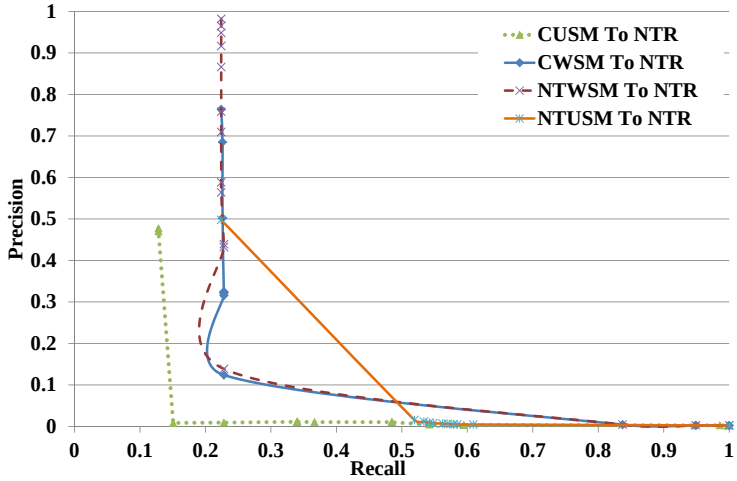


Fig. 3.8: Precision-recall curves for different models tested on NTR

STUSM. Fig. 3.9 shows the PR- curves of these two models with the *STR* and both of them produces very similar curves and for the higher value of recall at some points *CWSM* produces higher precision. So, we can conclude from the AUCs and PR-curves that *CWSM* is the best model for the *STR*. For the *LTR*, it is clear that the data must be re-balanced before learning for this type of cases. Fig. 3.10 shows the PR curves for *CUSM*, *TUSM*, and *LTUSM* when applied to the *LTR*. It shows that *TUSM* produces the best PR curve. So, from the AUCs and PR curves we can conclude that *TUSM* is the most appropriate model for the *LTR*.

The fragmentation helps to build specialized models; however, it also reduces the training data size. To see the effect of training data size on the AUC, we learned decision tree models with the *CR* (without re-balancing) with different data size and the results are reported in Fig. 3.11. It shows that for lower numbers of training data set like 20K and 40K the AUCs are low and with increase in the number of training examples it becomes stable at 0.88.

From the comprehensive experiments, we can conclude that for achieving an unbiased and good ranking the training data may need to be re-balanced before applying the data mining tasks. In our data set the decision tree C4.5 algorithm is the most appropriate choice for classification and ranking. In our case, re-balancing with under-sampling helps us to achieve a better model for the transit bags. We also learn that for a better ranking, it may require learning specialized models for different groups of data in the whole data set like we did for non-transit, transit, shorter and longer transit data. In our data set for the non-transit records, longer transit records, and shorter transit records

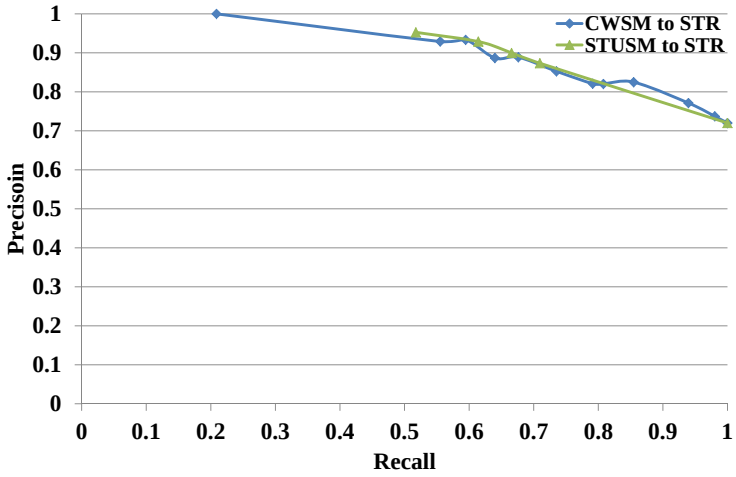


Fig. 3.9: Precision-recall curves for different models tested on STR

the chosen models are *CWSM, TUSM,* and *CWSM* respectively. Moreover, it is also learned that a larger number of training records also important for a better and stable ranking.

Extracted Rules and Analysis: We now explore the patterns found for the different fragments by their chosen classifier and the assigned probability estimate scores (risk scores) of each pattern. For each fragment, we report the top 5 rules with the highest risk scores followed by analysis. For reasons of confidentiality, the airport names in a pattern have been changed to A1, A2, ..., A6.

***For NTR by CWSM*:**

Rule1: *If DurationBeforeFlight*$\leq$*2min* $\Rightarrow$ *MH* [Score: 0.98]

Rule2: *If DurationBeforeFlight*>*2min AND FromAirport=A8 AND IsLongerStayFound=1 AND TotalBagInThatHour*> 192 $\Rightarrow$ *MH* [Score: 0.88]

Rule3: *If DurationBeforeFlight*>*2min AND FromAirport=A8 AND IsLongerStayFound=1 AND TotalBagInThatHour*$\leq$*192 AND DurationBeforeFlight*$\leq$*65min* $\Rightarrow$ *MH* [Score: 0.66]

Rule4: *If DurationBeforeFlight*>*2min AND FromAirport=A8 AND IsLongerStayFound=1 AND TotalBagInThatHour*$\leq$*192 AND DurationBeforeFlight*>*65min* $\Rightarrow$ *OK* [Score: 0.27]

Rule5: *If DurationBeforeFlight*>*2min AND FromAirport=A2 AND DurationBeforeFlight-Flight* >*40min* $\Rightarrow$ *OK* [Score: 0.12]

***For STR by CWSM*:**

Rule1: *If DurationBeforeFlight*$\leq$*2min* $\Rightarrow$ *MH* [Score: 0.98]

Rule2: *If DurationBeforeFlight*>*2min AND FromAirport=A2 AND DurationBeforeFlight* $\leq$*25min* $\Rightarrow$ *MH* [Score: 0.93]

Rule3: *If DurationBeforeFlight*>*2min AND FromAirport=A3 AND DurationBeforeFlight* $\leq$ *9min* $\Rightarrow$ *MH* [Score: 0.91]

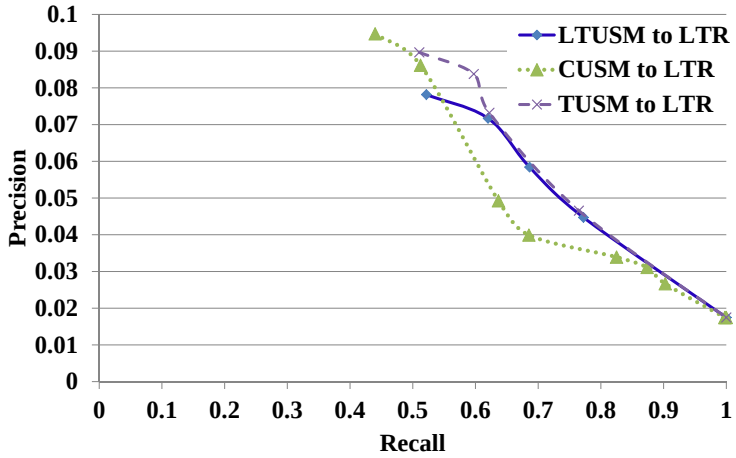


Fig. 3.10: Precision-recall curves for different models tested on LTR

Rule4: *If DurationBeforeFlight>25min AND FromAirport=A2 AND TotalBagInThatHour> 145 ⇒ MH* [Score: 0.71]
Rule5: *If DurationBeforeFlight > 29min AND FromAirport=A2 AND IsLongerStayFound=1 AND TotalBagInThatHour>115 AND DurationBeforeFlight ≤ 35min ⇒ MH* [Score: 0.63]
***For LTR by TUSM*:**
Rule1: *If DurationBeforeFlight≤54min ⇒ MH* [Score: 0.88]
Rule2: *If DurationBeforeFlight>54min AND IsLongerStayFound=1 AND DurationBeforeFlight≤75min ⇒ MH* [Score: 0.64]
Rule3: *If DurationBeforeFlight>75min AND IsLongerStayFound=1 AND DurationBeforeFlight ≤ 95min ⇒ OK* [Score: 0.45]
Rule4: *If DurationBeforeFlight>95min AND IsLongerStayFound=1 ⇒ OK* [Score: 0.28]
Rule5: *If DurationBeforeFlight>54min AND IsLongerStayFound=0 ⇒ OK* [Score: 0.18]

The above rules show that available baggage handling time before the flight departure is always an important issue regardless of the category of the bag. For the non-transit bags the departure airport is a very important factor and based on the *FromAirport* the other risk factors like check-in time of the bag before the flight, longer stay between locations, and total number of bags during the flight hour have high influence on the baggage management problem. Rules 4 and 5 of the *NTR* can be discarded as they have very low risk scores. For the *STR*, it is considered to be mishandled by default. The risk factors and the effect of transition duration for the *STR* also vary based the transit airport. The rules show that when the transit duration increases, the other factors like a longer stay between baggage handling locations and number of bags during the flight hour take influence on the baggage management problem. In case of longer transit records, a record with *DurationBeforeFlight ≤ 54min* is directly classified as *MH* regardless of any other condition.

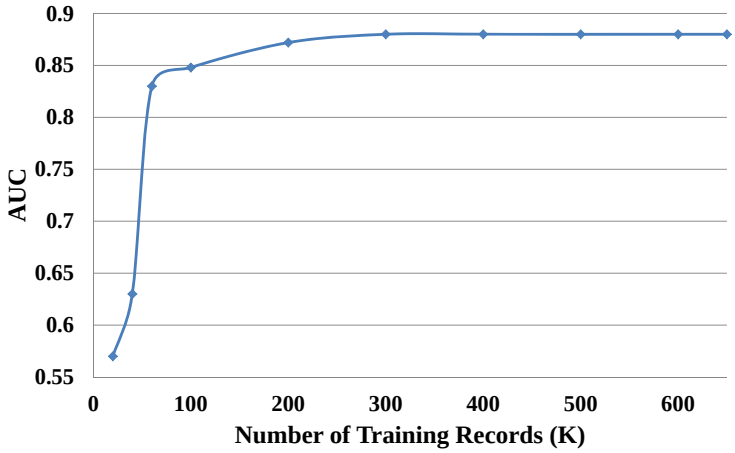


Fig. 3.11: AUC with different size of training data

This condition also reflects Fig. 3.6 discussed earlier, where the mishandling rate suddenly started increasing fast around this point, and it is almost 100% when the duration $\leq$ 31min. The rules of *LTR* also show that if *DurationBeforeFlight* $>$ *54min* then a longer stay at a location is highly responsible for baggage mishandling. The rules 3, 4, and 5 of the *LTR* can be discarded due to their low risk scores.

5 Related Work

Related work falls into two main categories. One is to pre-process unstructured RFID-based tracking data, and another one is to perform data mining task on imbalanced data set. Warehousing and mining techniques of RFID data from supply chain systems have been proposed in [41]. They convert the raw RFID records into cleansed record containing the first and last reading times of an object under the readers activation range. They took the advantage of bulky movement of objects for compressing the huge volume of RFID data. A data warehouse for analyzing RFID-based baggage tracking data is proposed in [15], where the raw tracking records are converted into *StayRecords* along with other dimensions. In [14, 16] the raw reading records are converted into mapping records containing the entry and exit times of an object at a constrained (e.g., conveyor belts of airport baggage management) and semi-constrained indoor symbolic locations (e.g., large hall, rooms, etc.,). In the present chapter, we further refine the *stay records* into *FlightLeg records* that capture different aggregate information from the stay records including other dimensional information for a higher level analysis.

Several papers address the problems of mining with imbalanced data

set [28, 59]. The main approaches of dealing with imbalanced data are re-sampling (includes under-sampling [56] and over-sampling [28]) and cost-sensitive learning [30]. Measuring the performance of classifiers and comparing models specially for imbalanced data set scenario have been discussed widely [28, 59]. We use AUC as the main measure [24, 43, 56] for comparing the models as well as present precision-recall curve as there is a deep relation between ROC and PR space [29]. In the present chapter, we apply several data mining techniques with different re-balancing techniques for finding best classifier and re-balancing techniques that can provide a good ranking in our data set.

6 Conclusion and Future work

In this chapter, we proposed a detailed methodology for finding risk factors from the imbalanced RFID airport baggage tracking data. We presented the pre-processing steps for preparing the raw RFID tracking data into *FlightLeg records*. We estimated the risk score of a bag being mishandled. In order to compute the risk scores, we learned classifiers that assigned scores and then evaluated the quality of the scores with the AUC measure. We dealt with the imbalance problem, applied different data mining techniques, and based on AUCs and Precision-Recall curves we found that the decision tree is the best classifier for our data set. We fragmented the data set into transit, non-transit, shorter and longer transit and obtained the appropriate models for the different fragments. We also found that re-balancing the data set by under-sampling helps to achieve a better predictive model for the longer transit bags. We conducted comprehensive experiments with real baggage tracking data, and it showed that fragmenting and mining each of the fragments separately was a right choice. The extracted patterns show that overall available handling time for a bag is a critical factor and; more specifically, a bag is considered to be a high risk if it has less than 54 minutes in the transit airport. For non-transit bags, the factors depend on the departure airport. It was also found that a longer stay between baggage handling locations and the total number of bags during the flight hour are important factors to predict mishandling as well. The proposed methodology can help the aviation industry with examining baggage management problems for further improvement in the system.

Several directions for future work exist. First, a more thorough study of the root causes for mishandling, which is non-trivial, given the low probability of Mishandled events. Second, analyzing baggage handling sequences for finding problems in the system. Third, finding spatio-temporal outliers from the RFID baggage tracking data. Fourth, developing native support from the data mining tools like automatic methods for finding the most appropriate

models.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

Chapter 4

Onine Risk Prediction for Indoor Moving Objects

The paper has been accepted in the
Proceedings of the IEEE 17th International Conference on Mobile Data Management(IEEE MDM 2016), Porto, Portugal, 2016. The layout of the paper has been revised.

Abstract

Technologies such as RFID and Bluetooth have received considerable attention for tracking indoor moving objects. In a time critical indoor tracking scenario such as airport baggage handling, a bag has to move through a sequence of locations until it is loaded into the aircraft. Inefficiency or inaccuracy at any step can make the bag risky, i.e., the bag may be delayed at the airport or sent to a wrong airport. In this chapter, we propose a novel probabilistic approach for predicting the risk of an indoor moving object in real-time. We propose a probabilistic flow graph (PFG) and an aggregated probabilistic flow graph (APFG) that capture the historical object transitions and the durations of the transitions. In the graphs, the probabilistic information is stored in a set of histograms. Then we use the flow graphs for obtaining a risk score of an online object and use it for predicting its riskiness. The chapter reports a comprehensive experimental study with multiple synthetic data sets and a real baggage tracking data set. The experimental results show that the proposed method can identify the

risky objects very accurately when they approach the bottleneck locations on their paths and can significantly reduce the operation cost.

1 Introduction

Technologies such as RFID and Bluetooth enable a variety of indoor, outdoor, and mixed indoor-outdoor tracking applications. Examples of such applications include tracking people's movement in large indoor spaces (e.g., airport, office building, and shopping malls), airport baggage tracking, item movement tracking in supply chains, and package tracking in logistics systems. During the movement of the objects, these tracking applications record the symbolic locations of the objects at different time points. For example, consider an RFID baggage tracking application where RFID readers are deployed at the different baggage handling locations such as check-in, screening, sorter, etc. Each reader has a very limited tracking range that covers a small portion of the location. If an object containing an RFID tag moves from check-in to the sorter, this produces two consecutive tracking records of the object location in different places. Due to limitations in indoor positioning technologies, the locations between these two records are not obtained. We call this type of tracking Symbolic Location Tracking (SLT). An SLT system can generate a massive volume of tracking data. This massive tracking data can be very useful for analyses such as finding risk factors, problem discovery, and decision-making. For example, in an airport baggage handling system, a bag can be left behind in the airport (i.e., failed to catch the intended flight) or can be sent to a wrong airport. In the baggage tracking system, the baggage tracking data can be used to extract interesting patterns and find the reasons for baggage mishandling. In a supply chain system, the item tracking data can be used for finding the factors that lead items being returned or getting rot. Some work has been carried out for the efficient management of such tracking data and to analyze them in the offline scenario [16, 36]. However, using such data for time critical online applications, such as online bags risk prediction in the airports can be very useful for getting real-time notifications for immediate handling of the risky bags. Moreover, online items risk discovery in supply chain and production systems, online item risk prediction in logistics systems, traffic jam prediction, etc., can also benefit from the insights obtained from analyzing the online data. An example of a real-time analysis request can be: "notify the baggage management team whenever a bag becomes risky during its processing time at Aalborg airport". Another request can be: "which are the 5 current bags with the highest risk of not reaching their plane on time?".

The chapter has several contributions. First, to the best of our knowledge this is the first chapter to propose a method for online risk prediction for indoor moving objects. Second, we propose the concepts of least duration probability (LDP), aggregated LDP (ALDP), LDP histogram (LDPH), and ALDP histogram (ALDPH) where the histograms store probabilistic in-

formation about the transition times of the historical objects. We propose a probabilistic flow graph (PFG) and aggregated PFG (APFG) that capture the flows of objects from one symbolic location to another and the edges of the graphs contain corresponding LDPH and ALDPH, respectively. Third, an online risk prediction (ORP) algorithm is proposed that uses the PFG and APFG for obtaining a risk score for an online indoor moving object. The risk score is used for predicting the riskiness of the object during its processing. Fourth, as the total available processing time of an object is an important factor, we propose an approach for normalizing the available processing time with the stay durations of objects at different locations for obtaining a better risk score. Fifth, we present a cost model for obtaining the best risk score threshold that can maximize the overall benefit of identifying and removing the risky objects. Sixth, the chapter reports a comprehensive experimental study with several synthetic data sets following different data distributions and a real baggage tracking data set. The results show that the proposed method can produce a very accurate risk score and identify the risky objects very precisely in different types of data distributions.

The remainder of the chapter is organized as follows. Section 2 presents the SLT systems and tracking data. Section 3 discusses the problem formulation. Section 4 presents the solution and probabilistic flow graph. Section 5 presents online risk prediction steps and the algorithm. Section 6 reports the experimental results. Section 7 reviews related work. Section 8 concludes and points to future work.

2 Preliminaries

SLT Systems. In an SLT system, tracking devices are strategically deployed at different fixed symbolic locations, such as different doors in an office space, between sections in an airport, different locations in airport baggage management, etc. The objects contain tags or devices that can be tracked by the tracking devices. For example, in the case of RFID technology, RFID readers and RFID tags are used; in the case of Bluetooth systems, Bluetooth access points and Bluetooth devices are used. After deployment of the tracking devices, the positions are recorded in the database.

Fig. 4.1 shows an example of airport baggage tracking scenario. The upper part of the figure shows the top level path of a bag that traveling from Aalborg Airport (AAL) to Brussels Airport (BRU) via Copenhagen Airport (CPH). The bag has to go through several baggage processing steps inside each airport. The bottom part of the figure shows the baggage processing stages inside AAL. The circles represent the baggage tracking locations where RFID readers are deployed for baggage tracking. Before handing over a bag into the system, an RFID tag with some encoded information about the bag

and the route is attached to the bag. Suppose the bag is intended for *Flight1* and the preplanned path for the bag is: "*check-in* → *Screening*→ *Sorter1* → *Gateway1* → *BeltLoader1*". Mismanagement or inefficiency at any one of these transitions may result in the bag being mishandled, i.e., the bag might miss the flight due to delay, or the bag might be sent to wrong flight. While passing through the different locations, the bag enters the activation range of an RFID reader, it is continuously detected by the reader with a sampling rate, and it generates *raw reading records* with the form: $\langle obj, Loc, t \rangle$. It means that a reader placed at location *Loc* detects a moving object *Obj* in its activation range at time *t*. An example set of raw reading records in an *SLT* system is shown in Table 4.1.

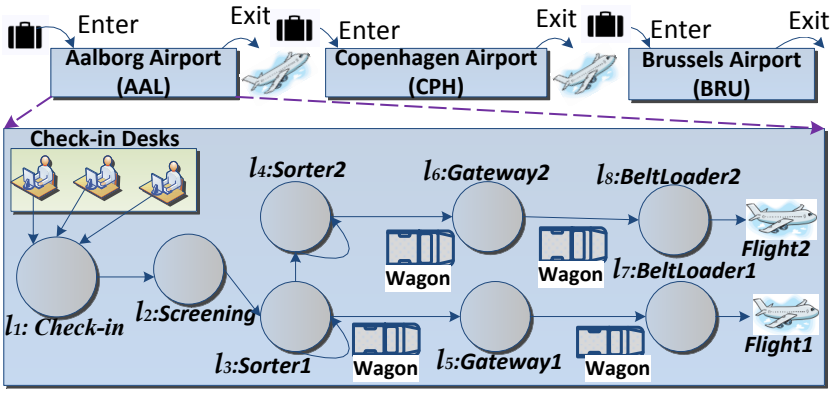


Fig. 4.1: Example SLT scenario in airport baggage tracking

Table 4.1: Raw Tracking Data

Obj	$\langle$ *Obj, Loc, t* $\rangle$
o_1	$(o_1, l_1, 1)$ $(o_1, l_1, 3)$ $(o_1, l_1, 5)$ $(o_1, l_2, 12)$ $(o_1, l_2, 14)$ $(o_1, l_3, 25)$ $(o_1, l_3, 27)$ ...
o_1	$(o_2, l_1, 10)$ $(o_2, l_1, 12)$ $(o_2, l_2, 26)$ $o_2, l_3, 32)$ $(o_2, l_4, 39)$ $(o_2, l_4, 41)$ $(o_2, l_6, 46)$ $(o_2, l_8, 55)$ ...
...	...
o_{1000}	...

StayRecords. As seen, the raw readings contain many redundant records. If an object stays for *t* time units under the activation range of a tracking device, it can generate *t/sampling rate* number of records for that stay. De-

pending on the application scenario, the stay of an object under the activation range of a reader can vary. For example, in an airport baggage tracking scenario a bag continuously moves from one location to another and usually it stays very short period under the activation range. Whereas, in a supply chain scenario an object can stay long (e.g., few hours, days) on a shelf and can stay for a long period under a reader. However, in any SLT application, an object moves from one symbolic location to another, and it is essential to know the total duration spent by the objects between the locations. We create a table *StayRecord*$\langle Obj, L_{from}, L_{to}, t_s, t_e, Dur\rangle$, which represents that an object *Obj* first appeared at location L_{from} at time t_s and then first appeared at the next location L_{to} at time t_e. It took *Dur* time to go from the reader at L_{from} to the reader at L_{to} or in another way it spent *Dur* time between L_{from} and L_{to}. Table 4.2 shows an example of *StayRecord* table constructed for the raw reading records shown in Table 4.1.

Table 4.2: Stay records from Table 4.1

Obj	*StayRecord*$\langle Obj, L_{from}, L_{to}, t_s, t_e, Dur\rangle$
o_1	$(o_1, l_1, l_2, 1, 12, 11)$ $(o_1, l_2, l_3, 12, 25, 13)$
o_2	$(o_2, l_1, l_2, 10, 26, 16)$ $(o_2, l_2, l_3, 26, 32, 6)$ $(o_2, l_3, l_4, 32, 39, 7)$ $(o_2, l_4, l_6, 39, 46, 7)$ $(o_2, l_6, l_8, 46, 55, 9)$
...	...
o_{1000}	$(o_{1000}, ..., .., ..., ..., ...)$ ...

3 Problem Formulation

We consider an application scenario where an object can be processed in a single system or multiple subsystems throughout its journey from the origin to the final destination. In the case of subsystems, when an object is registered in its origin, its identifier and the global route are shared within all the subsystems for further processing. Depending on the application scenario, an online risk prediction system can monitor the object throughout its entire journey from origin to final destination or it can monitor the object individually within each subsystem between its entry and exit times within that subsystem. For example, in Fig. 4.1, the global path of the object is *AAL*→*CPH*→*BRU*. The overall processing of the bag should be processed by the three subsystems, i.e., first at AAL, second at CPH, and third at BRU. Whenever the bag first registered at AAL, its identifier, route and flight information is shared to CPH and BRU, so that they can recognize the bag when it appears to their systems. In this context, each subsystem has its separate online risk prediction system. Whenever a bag first detected by an RFID

reader at AAL, the bag become online to the local risk prediction system and it starts monitoring the bag until it exits AAL, or until it is confirmed that the bag missed its flight. Similarly, when the bag is detected at CPH, it becomes online at CPH and so on.

Definition 1. ***Online Object.*** An object is considered as an *online object* to a system/subsystem at time t, if t falls in the time interval $[t_{enter}, t_{exit}]$, where t_{enter} is the first time the object tracked in a tracking device in the system/subsystem and t_{exit} is the last time the object tracked by the last tracking device in the system/subsystem or the time within which the object is expected to exit the system/subsystem.

Problem Statement. Given a set of stay records R and a set of online moving objects O, we are interested in building a predictive model from R that can predict, as early as possible, whether an object $o_i \in O$ is at risk in real-time.

For example, in baggage tracking, the model should be able to predict whether a bag going through the baggage handling stages is at risk of being delayed at the airport and the prediction should be made as early as it sees the bag is being abnormally differed compared to other bags.

4 Solution

4.1 Solution Outline

The overall outline of the data collection and risk prediction steps is shown in Fig. 4.2. The online object tracking data stream is passed into two sections. One of them stores the data offline for future analysis and model building purpose and another uses it during the online risk prediction process. The offline/historical reading records are processed and converted into *StayRecords*. The stored *StayRecords* are used for building the probabilistic model. The model, raw data stream and the preplanned path of the objects are used by the *Online Risk Prediction(ORP)* for deciding which objects are at risk. Finally, risky objects are notified by the ORP for special handling.

Let, L = $\{l_1, l_2, l_3, ..., l_n\}$ be the set of locations available in the data set. A set of durations taken by the transitions from location l_i to l_j be $D_{i,j}$=$\{d_1, d_2, d_3, ... , d_n\}$.

Definition 2. ***Least Duration Probability (LDP).*** A least duration probability (LDP) for a movement l_i to l_j with threshold duration $d_k \in D_{i,j}$ is defined as,

$$LDP(l_i, l_j, d_k^{\geqslant}) = \frac{Count(l_i, l_j, d_k^{\geqslant})}{Count(l_i, l_j)} \tag{4.1}$$

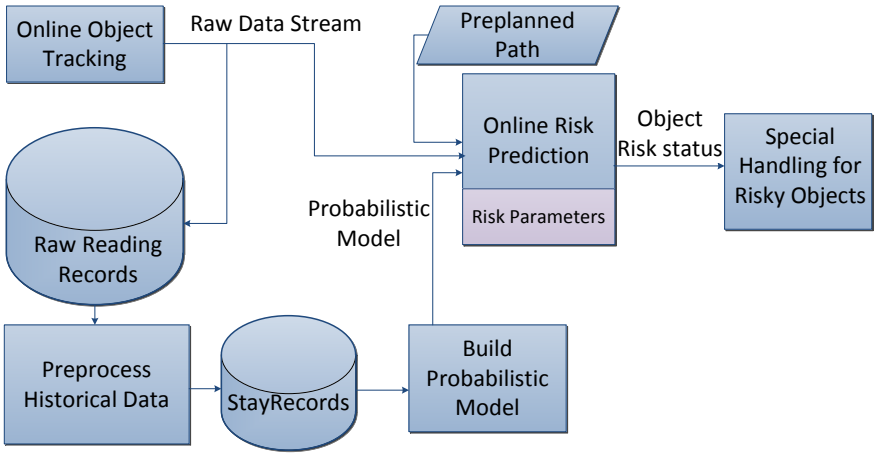


Fig. 4.2: Outline of the overall system

In Eq. (4.1), $Count(l_i, l_j, d_k^{\geq})$ = total number of objects that took at least d_k duration between l_i to l_j and $Count(l_i, l_j)$ = total number of objects that have a transition from l_i to l_j.

Definition 3. ***Least Duration Probability Histogram (LDPH).*** A least duration probability histogram (LDPH) for transitions between l_i to l_j is a histogram with transition durations $D_{i,j}$ on the X-axis, and LDPs for the transitions on the Y-axis.

Table 4.3 shows an example summary of transitions from the stay records in Table 4.2. Fig 4.3 shows the different LDPHs for the transitions shown in Table 4.3. For example in Fig. 4.3a, LDP=0.7 represents that the probability of transition from l_1 to l_2 with a duration ≥ 16 is 0.7. The figure also shows that the LDP for duration 28 is very low (0.02).

4.2 Probabilistic Flow Graph (PFG)

We use a probabilistic flow graph (PFG) for modeling the movement of objects from one symbolic location to another. The PFG is formally defined by a labeled directed graph $G = (L, E, D, H, lb_E)$, where:

1. L is the set of locations where each location is represented as a vertex in G.

2. E is the set of directed edges: $E = \{(l_i,l_j) \mid l_i, l_j \in L\}$.

Table 4.3: Transition Summary (C stands for *Count*)

Transition tr **(**$l_i \rightarrow l_j$**)**	**Dur (**d_k**)**	$C(l_i)$	$C(tr)$	$C(tr, d_k^{\geqslant})$	$LDP(tr, d_k^{\geqslant})$
$l_1 \rightarrow l_2$	13 16 20 28	1000	1000	1000 700 300 20	1 0.7 0.3 0.02
$l_2 \rightarrow l_3$	8 10 17	1000	1000	1000 580 50	1 0.58 0.05
$l_3 \rightarrow l_4$	50 65 95	1000	500	500 250 25	1 0.5 0.05
$l_3 \rightarrow l_5$	60 70 98	1000	470	470 250 50	1 0.53 0.11
$l_4 \rightarrow l_6$	70 75 99	500	480	480 280 30	1 0.58 0.06
$l_5 \rightarrow l_7$	40 50	470	460	460 250	1 0.54
$l_6 \rightarrow l_8$	45 48	480	455	455 200	1 0.44

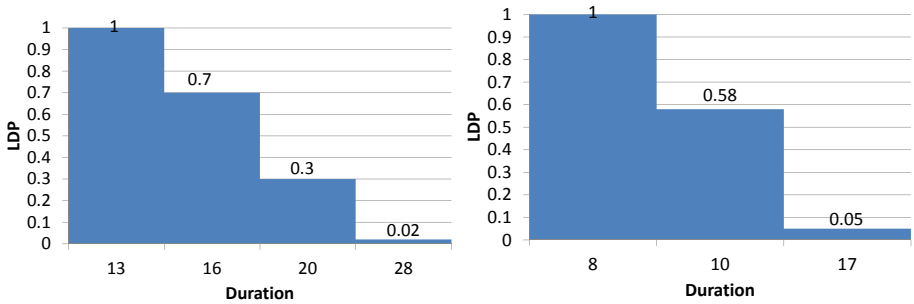
3. D is the set of durations. $D_{i,j} \subseteq D$ represents the set of durations taken by objects for the transitions from l_i to l_j.

4. H is the set of LDPHs, where an $LDPH_{i,j} \in H$ is computed from the number of transitions from location l_i to l_j and the durations $D_{i,j} \subseteq D$.

5. lb_E is a function lb_E: $E \rightarrow H$ that labels an edge by an LDPH, $h \in H$. An edge $(l_i, l_j) \in E$ is labeled by an LDPH $LDPH_{i,j} \in H$, where $LDPH_{i,j}$ is the LDPH from l_i to l_j.

Fig. 4.4 shows the PFG constructed from the transition summary shown in Table 4.3. Two LDPHs of Fig. 4.4 is shown in Fig 4.3. However, all the data for rest of the LDPHs are available in the $LDP(tr, d_k^{\geqslant})$ column of Table 4.3.

5 Online Risk Prediction (ORP)



(a) LDPH for $l_1 \rightarrow l_2$ **(b)** LDPH for $l_2 \rightarrow l_3$

Fig. 4.3: LDPHs for the transitions $l_1 \rightarrow l_2$ and $l_2 \rightarrow l_3$ in Table 4.3

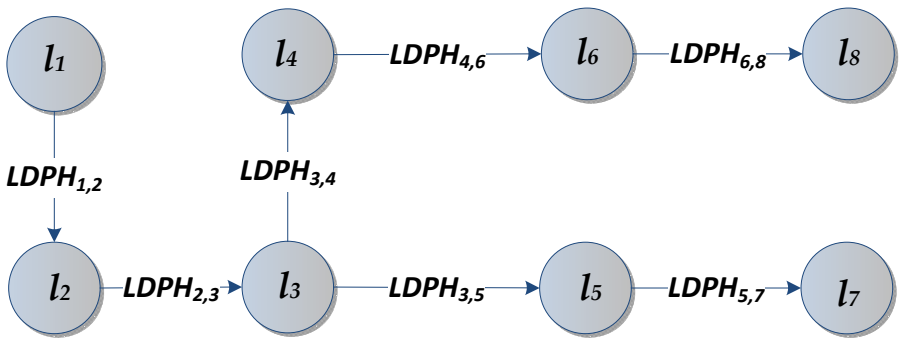


Fig. 4.4: Probabilistic flow graph (PFG)

5.1 ORP Overview

In an *SLT* system, the movements of the mishandled objects are expected to differ from the usual movement. They can take a wrong transition or can stay longer between planned locations. As the PFG is learned from the historical data, we use it for obtaining a probability score of an online object and use the score for predicting the unusual movements.

We consider a scenario, where the path of a given online object is predefined. For example, in case of the baggage tracking, all the bags intended for a particular flight *SK123* should follow the same path sequence starting from the check-in desk up to the belt loader to the aircraft (e.g., $l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_5 \rightarrow l_7$). If the object does not follow its preplanned path, it is triggered as risky. However, an object following its preplanned path, but unusual longer duration for a transition can make the object risky. We use two different thresholds to decide the riskiness of an object. The method of finding the threshold is discussed at the end of this section. For each transition $l_i \rightarrow l_j$ in the PFG, we use an LDP threshold $LDP_{th}(l_i, l_j)$, that helps to get the maximum acceptable

stay duration ($Dur_{max}(l_i, l_j)$) of an object between l_i and l_j. If an object spends equal or more than $Dur_{max}(l_i, l_j)$ between l_i and l_j, the object is considered at risk. Another threshold is called *risk score threshold (RS_{th})*. After each transition of an object o, its combined duration probability (CDP) for the so far traversed path is computed by Eq. (4.2). In Eq. (4.2), n is the total number of transitions of o and LDP'_i is the LDP for the stay duration of o obtained from the LDPH for o's i_{th} transition. The value of CDP is converted into *risk score (RS)* by, *RS = 1-CDP*. If $RS \geq RS_{th}$, we trigger that o is at risk. Maintaining the value of *RS* helps to find the top-k risky online objects in the system.

$$CDP(o) = \prod_{i=1}^{n} LDP'_i \tag{4.2}$$

Generally, when a PFG is learned from a large data set, the LDPHs should contain most of the possible stay durations for the upcoming new objects. However, if a new object o takes d duration for a transition $l_i \rightarrow l_j$ and d is not directly available in LDPH(l_i,l_j), the value of LDP (l_i, l_j, $d^{\geqslant}$) is computed in one of the following ways:

- If $d < d_{first}$ (the first entry of LDPH(l_i,l_j)), then LDP (l_i, l_j, $d^{\geqslant}$) = LDP(l_i, l_j, $d^{\geqslant}_{first}$).
- If $d > d_{last}$ (the last entry of LDPH(l_i,l_j)), then LDP (l_i, l_j, $d^{\geqslant}$) = LDP(l_i, l_j, $d^{\geqslant}_{last}$).
- If $d_{\sqsubset}$ and $d_{\sqsupset}$ are two consecutive entries in LDPH(l_i,l_j) and $d_{\sqsubset} < d < d_{\sqsupset}$, the value of *LDP ($l_i$, l_j, $d^{\geqslant}$)* is computed by the linear interpolation of the LDPs as shown in Eq. (4.3). The equation computes the rate of decrease in the LDP between durations $d_{\sqsubset}$ and $d_{\sqsupset}$ and then calculates the total expected probability change for d from the distance between d and $d_{\sqsubset}$. After that the expected probability change is subtracted from *LDP($l_i, l_j, d^{\geqslant}_{\sqsubset}$)* to get the expected value of *LDP (l_i, l_j, $d^{\geqslant}$)*.

$$LDP'(l_i, l_j, d^{\geqslant}) = LDP(l_i, l_j, d^{\geqslant}_{\sqsubset}) - \frac{LDP(l_i, l_j, d^{\geqslant}_{\sqsubset}) - LDP(l_i, l_j, d^{\geqslant}_{\sqsupset})}{d_{\sqsupset} - d_{\sqsubset}} \times (d - d_{\sqsubset}) \tag{4.3}$$

Furthermore, as the new online object becomes part of the historical data after its operation, its new duration is included in the PFG next time when a new model is built. Now coming to the ORP, if $LDP_{th}(l_i, l_j)$ is not directly available from LDPH(l_i, l_j), we use Eq. (4.4) for computing the Dur_{max} for that transition. Eq. (4.4) is derived from Eq. (4.3). The equations are mapped by considering, $d = Dur_{max}(l_i, l_j)$ and $LDP(l_i, l_j, d^{\geqslant}) = LDP_{th}(l_i, l_j)$. The durations $d_{\sqsubset}$ and $d_{\sqsupset}$ are retrieved from LDPH(l_i, l_j) where they are

the keys for two consecutive LDPs $LDP_{\sqsubset}$ and $LDP_{\sqsupset}$, respectively (where $LDP_{\sqsubset}<LDP_{th}(l_i,l_j)<LDP_{\sqsupset}$).

$$Dur_{max}(l_i,l_j) = \left\lceil d_{\sqsubset} + \frac{LDP(l_i,l_j,d_{\sqsubset}^{\geqslant}) - LDP_{th}(l_i,l_j)}{LDP(l_i,l_j,d_{\sqsubset}^{\geqslant}) - LDP(l_i,l_j,d_{\sqsupset}^{\geqslant})} \times (d_{\sqsupset} - d_{\sqsubset}) \right\rceil \quad (4.4)$$

For example, consider an object o following a path: $l_1 \xrightarrow{17} l_2 \xrightarrow{17} l_3 \xrightarrow{60} l_5 \xrightarrow{40} l_7$. The labels in the arrows represent the duration taken for the transitions. Let us consider that the object followed its preplanned path. As the LDPH(l_1,l_2) has no entry for 17, its expected value by Eq. (4.3) is, LDP (l_1,l_2,$17^{\geqslant}$) = 0.7-(0.7-0.3)/(20-16)×(17-16) = 0.6. The full CDP for the object = LDP(l_1, l_2, $17^{\geqslant}$) × LDP(l_2, l_3, $17^{\geqslant}$) × LDP(l_3, l_5, $60^{\geqslant}$) × LDP(l_5, l_7, $40^{\geqslant}$) = 0.6×0.05×1×1 = 0.03. Let us consider that, RS_{th} = 0.8 and LDP_{th} for each of the transitions is also 0.2. So, $Dur_{max}(l_1,l_2)$ by Eq (4.4) = $\lceil 20 + (0.3-0.2)/(0.3-0.02) \times (28-20) \rceil$ = 23. Similarly, $Dur_{max}(l_2,l_3)$=15. When o completes its first transition (i.e., $l_1 \xrightarrow{17} l_2$), it passes both of the LDP and RS checks for that transition as the spent duration 17<Dur_{max} = 23 and RS = 1-CDP = 1-0.6 = 0.4$\ngeq RS_{th}$=0.8. So, the bag is not risky until the current state. When o reaches at l_3 (i.e., $l_2 \xrightarrow{17} l_3$), the spent duration 17$\nless Dur_{max}$ = 15. Furthermore, the CDP of o up to this location is 0.03 (0.6×0.05). So, RS = 1-0.03 = 0.97$\geq RS_{th}$=0.8. So, o is considered as a risky object after this transition in terms of both Dur_{max} and RS_{th}.

5.2 ORP Steps

The overall processing steps of the ORP are shown in Fig. 4.5. The process continuously waits for new readings. When a new reading arrives, it checks whether the object o_i in the reading is new. If o_i is new, it is inserted into a hash table (HT) and its preplanned path is retrieved from the system. If o_i does not follow its preplanned path, it is marked as risky due to the wrong location. However, if the path is correct, its CDP is initialized to 1. Based on the LDP_{th} of the current location l_{curr} and the planned next location $l_{p_{next}}$, a time trigger $TT_{o_i}(o_i,t_{start},t_{e_{max}})$ is added with o_i, where t_{start} is the first reading time of o_i at l_{curr} and $t_{e_{max}}$=t_{start}+Dur_{max}. As mentioned earlier, the value of Dur_{max} is extracted from the corresponding LDPH. If o_i remains between l_{curr} and $l_{p_{next}}$ until the clock time reaches $t_{e_{max}}$, the time trigger TT_{o_i} is raised and the trigger mark o_i as risky. Coming back to the starting point, if the new reading contains an old object, the process checks whether the object has changed its location. If it is in the same location, the process continues waiting for a new reading and a raise of a time trigger. However, if the object changes its location, its planned location is checked and based on that its further processing such as CDP computation, RS checking, and time trigger

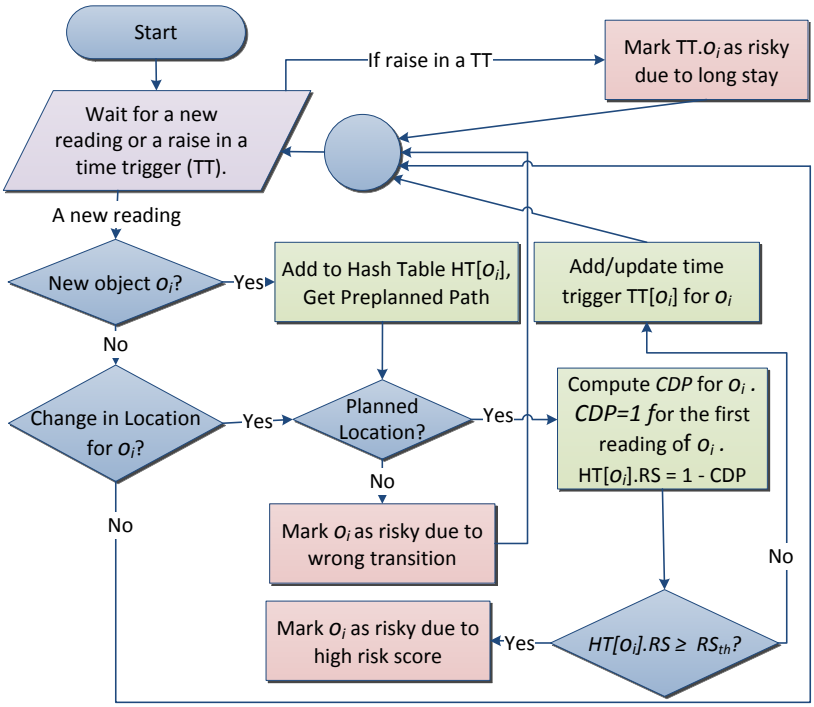


Fig. 4.5: Online risk prediction steps

update, etc., are performed. The time trigger allows a fast notification about a risky object as the process does not have to wait to complete the transition.

Algorithm 3 shows the processing of the ORP. It takes a hash table, RS_{th}, LDP_{th} list, and PFG as the input and update the hash table as the result. The algorithm continuously waits for a new reading or a raise of a time trigger (lines 1-2). If a new reading arrives, based on the data in the new reading, it updates the hash table HT. First, it checks whether the object in the new reading is newly arrived in the system (line 5). If the object is new, it is inserted in HT (line 6), and its preplanned path is checked (lines 6-9). If the object is in the planned location, it is initialized to a safe object (lines 11-12). Based on the next planned location and LDP_{th}, its Dur_{max} is extracted from the PFG (lines 13-14), maximum clock time threshold $t_{e_{max}}$ is calculated (line 15), and a time trigger is registered for the object (line 16). If the new reading does not contain a new object, it checks whether the object has changed its location (line 17). If the object has not changed its location, the algorithm continues waiting for a new reading or a raise in a time trigger. Conversely, if

the object changes its location in the new reading, its planned path is checked (lines 18-21). If it is in the planned path, its time trigger is updated with the new information (lines 22-25). After that, its stay duration for the transition is computed, CDP and RS are calculated, and the RS is checked with the RS_{th} (lines 26-31). Besides, if a time trigger is fired, the corresponding object is notified as risky (lines 33-35).

5.3 Recovery Scenario

The PFG cannot capture the possibility of a recovery of an object from its risky state. For example, an object might take long duration between location l_1 to l_2 that makes it risky. However, it might be handled very quickly in its next transition l_2 to l_3 that recovered the object from being mishandled. To capture this, we modify the PFG into *aggregate probability flow graph (APFG)*. Here, we additionally maintain an *aggregate LDPH (ALDPH)* for each path sequence S=$l_i l_{i+1} l_{i+3} ... l_n$, where l_i must be the first tracking location of at least one object in the data set and $i<n<=p$ (the length of the path sequence). An *ALDPH (S)* contains all the *aggregate LDPs (ALDP)* for S. An *ALDP(S,$d^{\geqslant}$)* represents the probability of taking at least d duration by an object for completing the path sequence S. The value of an ALDP for the path sequence S with a total duration d is computed by Eq. (4.5). In Eq. (4.5), $Count(S, d^{\geqslant})$ is the number of objects taking at least d duration to complete the path sequence S and $Count(S)$ is the number of objects that traveled through path S. Table 4.4 shows the ALDPs and data for ALDPHs for the path from l_1 to $l7$ in our example scenario.

$$ALDP(S, d^{\geqslant}) = \frac{Count(S, d^{\geqslant})}{Count(S)} \tag{4.5}$$

Table 4.4 shows the ALDPs and data for ALDPHs for our example scenario. As the table grows, we skip providing detail for some of the paths.

The processing of the ORP with the APFG is very similar to the algorithms discussed above with some additional conditions and operations. First, after each transition, in addition to the CDP computation, the ALDP for the traveled path is extracted from the corresponding ALDPH. If $CDP<ALDP$, then CDP is updated with the value of ALDP to make the score less risky. Also note that for the first transition the value ALDP and LDP is same. Second, instead of using LDP_{th} for each transition, we maintain an $ALDP_{th}(s_i)$ for each path sequence s_i for each of the preplanned paths. In our example scenario, for the path from l_1 to l_7, there will be ALDP thresholds for each of the path sequences mentioned in Table 4.4. Third, the concept of $Dur_{max}(l_i, l_j)$ is changed to $Dur_{max}(s_i)$, where Dur_{max} represents the maximum allowable duration for an object to complete the path sequence s_i. The value of $Dur_{max}(s_i)$

Algorithm 3: ORP(HashTable HT, RS_{th}, LDPThresholdList LDP_{th}, PFG)
Result: *Hash Table with Risk Status*

```
1  while true do
2      wait for a new reading or a raise in a time trigger;
3      if a new reading rr arrives then
4          o_i ← rr.Obj; l_cur = rr.Loc;
5          if HT[o_i] = NULL then
6              HT.Insert(o_i); PP[o_i]←PlannedPath(o_i);
7              if l_cur ≠ PP[o_i].POP() then
8                  HT[o_i].status ← "Risky";
9                  HT[o_i].Reason←"WrongTran"; continue;
10             HT[o_i].Loc_cur ← HT[o_i].Loc_prev ← rr.Loc;
11             HT[o_i].t_s←rr.t;HT[o_i].CDP←1;HT[o_i].RS←0;
12             HT[o_i].status←HT[o_i].Reason←"NotRisky";
13             l_pnext←HT[o_i].Loc_pNext←PP[o_i].POP();
14             Dur_max←PFG.GetDur(LDP_th, l_cur, l_pnext));
15             MaxTimeEnd t_emax ← rr.t+Dur_max;
16             TT[o_i] ← TimeTrigger(o_i, rr.t, t_emax);
17         else if HT[o_i].Loc_prev ≠ l_cur then
18             l_pcur ← PP[o_i].POP();
19             if l_cur ≠ l_pcur then
20                 HT[o_i].status ← "Risky"; HT[o_i].RS ← 1;
21                 HT[o_i].Reason←"WrongTran"; continue;
22             l_pnext←HT[o_i].Loc_pNext←PP[o_i].POP();
23             Dur_max←PFG.GetDur(LDP_th, l_cur, l_pnext));
24             MaxTimeEnd t_emax ← rr.t+Dur_max;
25             TT[o_i] ← TimeTrigger(o_i, rr.t, t_emax);
26             SpentDuration dur ← rr.t - HT[o_i].t_s;
27             HT[o_i].Loc_cur←l_cur; HT[o_i].t_s←rr.t;
28             HT[o_i].CDP←HT[o_i].CDP×LDP(HT[o_i].Loc_prev, l_cur, dur^≥);
               HT[o_i].RS ← 1- HT[o_i].CDP ;
29             if HT[o_i].RS ≥ RS_th then
30                 HT[o_i].status ← "Risky";
31                 HT[o_i].Reason ← "High RS";
32             HT[o_i].Loc_prev ← l_cur;
33     else if a time trigger is raised for the object o_j then
34         HT[o_j].status ← "Risky";
35         HT[o_j].Reason ← "Long Stay Triggered";
```

can be extracted from the $ALDPH(s_i)$ based on the $ALDP_{th}(s_i)$. Fourth, the concept of the time trigger is updated with the concept of ALDP and its

Table 4.4: Path Summary

Path (S)	**Dur** (d)	***Count***(S)	***Count***(S,$d^{\geqslant}$)	***ALDP***(S,$d^{\geqslant}$)
$l_1 \rightarrow l_2 \rightarrow l_3$	21	600	600	1
	23		500	0.83
	24		300	0.5
	30		150	0.25
	33		30	0.05
$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_5$	81	470	470	1
	83		415	0.88
	84		245	0.52
	94		195	0.41
	100		120	0.26
	121		50	0.11
	128		20	0.04
$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_5 \rightarrow l_7$	123	460	460	1
	131		290	0.63
	134		235	0.51
	144		145	0.32
	150		110	0.24
	171		40	0.09
	178		10	0.02
$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4$	74	100	100	1
	83		80	0.8
	86		50	0.5
	119		5	0.05
$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_6$	-	-	-	-
$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_6 \rightarrow l_8$	-	-	-	-
$l_1 \rightarrow l_2 \rightarrow l_4$	24	400	400	1
	26		385	0.96
	28		195	0.49
	30		170	0.42
	36		35	0.09
	38		25	0.06
	41		15	0.04
$l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_6$	-	-	-	-
$l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_6 \rightarrow l_8$	-	-	-	-

structure is changed to TT_{o_i} (s_i, $l_{p_{next}}$, t_{start}, $t_{e_{max}}$), where s_i is the so far completed path sequence by o_i, $l_{p_{next}}$ is the next planned location, t_{start} is the timestamp when o_i first tracked in the system, and $t_{e_{max}}$=t_{start}+$Dur_{max}(s_{p_{next}})$, where $s_{p_{next}}$ is the path sequence up to $l_{p_{next}}$ (i.e., $s_i \rightarrow l_{p_{next}}$). Then the rest of the procedure is same as the above algorithm.

Consider the example discussed above where an object o followed a path: $l_1 \xrightarrow{17} l_2 \xrightarrow{17} l_3 \xrightarrow{60} l_5 \xrightarrow{40} l_7$. In the above example, o was marked as risky when it completed the path up to l_3 (i.e., $l_1 \xrightarrow{17} l_2 \xrightarrow{17} l_3$). From Table 4.4, the ALDP up to l_3 is 0.05 (as the total duration=17 + 17 = 34). In terms of both LDP and ALDP, o is at risk at that point. After the next transition (i.e., up to l_5), the ALDP is 0.41, thus RS = 1- 0.41 = 0.59 $\ngeq RS_{th}$=0.8 (as total duration = 17 + 17 + 60 = 94). However, the value of CDP up to l_5 is 0.6 $\times$ 0.05 $\times$1 = 0.03. The value of CDP either decreases or remains same while multiplying new LDPs. The new score shows that the object recovered from its risky state as it was processed quickly between l_3 and l_5. So, we update the CDP with the value of ALDP and mark o as not risky. When o moves further, the time trigger for o is also updated based on the traversed path sequence, preplanned path, and $ALDP_{th}$.

5.4 Time Constrained ORP

Generally, a slow processing of an object at a location makes the object risky. This slow processing could also result in a dense location or traffic jam that could hamper the processing of the upcoming objects. However, there are many applications where an object has to reach a particular location within a given timestamp. For example, in the baggage tracking, a bag has to be loaded in the aircraft before the scheduled flight departure. So, the available duration before the flight departure is an important factor for baggage risk prediction. If a bag starts its processing well in advance before the flight departure, it is less risky, even if it stays longer for a transition. Conversely, a bag having a short duration before the flight makes it risky, even it is processed relatively quickly in its transitions. So, the stay duration should be normalized with the available processing time and use the normalized duration for taking the corresponding ALDP to reflect the actual riskiness of the object.

Let us consider, t_{enter} be the first time an object o detected in the system and t_{final} be the maximum timestamp when o should reach its final reading point/location. So, the total available duration for o is d_a=t_{final}-t_{enter}. The expected average duration of travel of an object is extracted from the *ALDPH* for the full preplanned path of the object. We consider d_e be that expected duration extracted from the *ALDPH* with *ALDP* = 0.5. After each transition of o, its normalized total stay duration for the so far traversed path is computed

by Eq. (4.6). In Eq. (4.6), dur_i is the stay duration of o for its i_{th} transition. In the equation, the value of *offset* is computed initially by subtracting the value of d_a from d_e. Then, after the k_{th} transition of o, its total travel time up to that transition is added to the offset for obtaining the normalized duration. So, instead of taking the ALDP directly for the total duration d_t, we take the ALDP for d_n. Depending on the value of d_a and d_e, the value of *offset* as well as d_n can be negative. As discussed earlier about picking the LDP from an LDPH for a given duration, the value of ALDP for d_n is also taken in the same way from the corresponding ALDPH.

$$d_n(o) = Offset + \sum_{i=1}^{k} dur_i, where\ offset = (d_e - d_a) \tag{4.6}$$

$O_1, O_2 : l_1 \xrightarrow{17} l_2 \xrightarrow{17} l_3 \xrightarrow{94} l_5 \xrightarrow{50} l_7$ $O_3 : l_1 \xrightarrow{17} l_2 \xrightarrow{17} l_3 \xrightarrow{49} l_5 \xrightarrow{40} l_7$

O_1: d_a = 200, *Offset* = 134-200 = -66

O_2: d_a = 144, *Offset* = 134-144 = -10

O_3: d_a = 100, *Offset* = 134-100 = 34

	$l_1 \rightarrow l_2$	$l_1 \rightarrow l_2 \rightarrow l_3$	$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_5$	$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_5 \rightarrow l_7$
O_1	d_n = -66+17 = - 49 ALDP = 0.6, $ALDP_n$ = 1	d_n = -66 + 34 = -32 ALDP = 0.05, $ALDP_n$ = 1	d_n = -66 + 128 = 62 ALDP = 0.04, $ALDP_n$ = 1	d_n = -66 + 178 = 112 ALDP = 0.02, $ALDP_n$ = 1
O_2	d_n = -10+17 = 7 ALDP = 0.6, $ALDP_n$ = 1	d_n = -10 + 34 = 24 ALDP = 0.05, $ALDP_n$ = 0.5	d_n = -10 + 128 = 118 ALDP = 0.04, $ALDP_n$ = 0.13	d_n = -10 + 178 = **168** ALDP = 0.02, $ALDP_n$ = 0.12
O_3	d_n = 34+17 = 51 ALDP = 0.6, $ALDP_n$ = 0.02	d_n = 34 + 34 = 68 ALDP = 0.05, $ALDP_n$ = 0.05	d_n = 34 + 83 = **117** ALDP = 0.88, $ALDP_n$ = 0.14	*Already delayed in the previous step*

Fig. 4.6: Examples of normalizing durations

For example, consider an object o_1 following its preplanned path and the stay durations for the transitions are: $l_1 \xrightarrow{17} l_2 \xrightarrow{17} l_3 \xrightarrow{94} l_5 \xrightarrow{50} l_7$. o_1 has a total of 200 seconds to reach l_7 from l_1. So, d_a=200 sec. From Table 4.4, d_e=134 (as ALDP for 134 is 0.51). So, *offset*=134-200 = -66. Now, for the first transition $l_1 \xrightarrow{17} l_2$, d_n = -66 + 17 = -49. So, from Table 4.3, the value of ALDP or LDP for the transition is 1. It shows that instead of taking the actual LDP for duration

17 (which was 0.6 as computed earlier), we take the LDP for normalized duration. As o has plenty of time to reach l_7, the normalization makes the object less risky. After the next transition to l_3, d_n = -66+17+17 = -32. So, the ALDP after normalization is 1. Before the normalization, the ALDP was 0.05. However, after normalization the score says that the object is completely safe until that transition. Similarly, when o_1 reaches at l_7, the total stay duration is 178 and the normalized duration is 112. Thus, before normalization the ALDP was 0.02 and after normalization ALDP is 1. It shows that, even o_1 takes long for its transitions, the normalization marks it as a safe object as it has a long available time to reach its destination. Fig. 4.6 shows the normalization process for 3 different objects with different values for d_a. It also shows the values of ALDPs before and after normalization ($ALDP_n$). The process for o_1 is already discussed. The figure shows that, the third transition of object o_3, gives very high ALDP (=0.88), whereas $ALDP_n$ is very low (=0.14). Thus, the normalization can better capture the actual riskiness of the object.

Adjusting Dur_{max} and Time Trigger. During processing of the ORP, $Dur_{max}(s_i)$ is adjusted to the concept of normalization. The normalized maximum allowable duration of an object o for completing its path sequence s_i is computed by, $Dur_{maxN}(s_i, o) = Dur_{max}(s_i) - offset$. As seen, if the value of *offset* is negative, then Dur_{maxN} allows more time to o_i. Whereas, the higher value of *offset* will reduce the value of Dur_{maxN} for adjusting the riskiness of o. Finally, $t_{e_{max}}$ in the corresponding time trigger is computed by, $t_{e_{max}} = t_{start} + Dur_{maxN}$ and is used for the risk prediction.

5.5 Finding the best thresholds

The optimal threshold depends on the particular goal of the system. We consider mishandled as a positive class for classification. A prediction system, giving too many false positives (FP) (i.e., predicting correctly handled objects as the mishandled objects) or false negatives (FN) (i.e., predicting mishandled objects as the correctly handled objects) can make the system useless or not interesting. So, there should be a defined acceptable metric for deciding the optimal operational threshold. We define a benefit function based on the operation cost, where the costs for the different kinds of errors are used for finding the threshold that maximizes the benefit. For example, In the case of baggage tracking, if a bag is predicted as mishandled, it requires a special manual handling so that the bag can reach the aircraft before the flight. If an FP occurs, there will be a waste in the human resource cost for the mistake. However, if an FN occurs, there will be a significant cost to deliver the bag to the passenger's address and as well as insurance and other operating costs are involved for such mistakes. So, in the baggage tracking scenario, the cost for an FN is much more compared to that for an FP. During model building and testing (discussed further in Section 6), we use Eq. (4.7)

for obtaining the total benefit for each of the generated thresholds and use the threshold that provides the maximum benefit. In Eq. (4.7), x= cost for handling a mishandled object (i.e., positive case (P)), y=cost for handling a predicted mishandled object (i.e., TP and FP), and #P= total number of positive cases in the data set. So, Eq. (4.7) can provide an idea how much money can be saved by using the ORP system.

$$Benefit(x, y) = x \times \#\mathrm{P} - (x \times \#\mathrm{FN} + y \times (\#\mathrm{TP} + \#\mathrm{FP})) \tag{4.7}$$

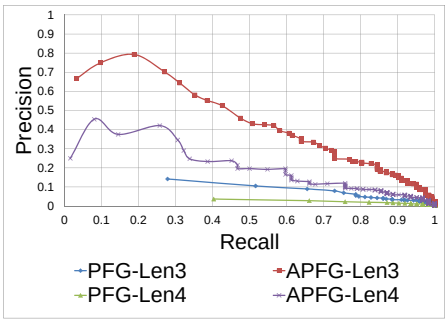
6 Experimental Evaluation

The PFG and APFG are implemented using a set of SQL statements and the prediction is implemented in C#. For all SQL queries, we use a leading RDBMS. The experiments are conducted on a laptop with an Intel Core i7 2.7 GHz processor with 8 GB RAM. The operating system is Windows 7 64 bit.

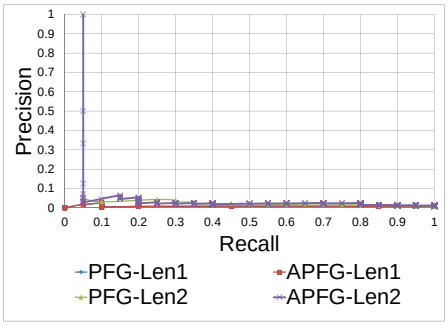
6.1 Data sets descriptions

We use both synthetic and real data for experimenting the different aspects of the proposed systems. The real data set reflects a specific scenario and contains a lot of erroneous readings, miss readings, and other anomalies. So, only experimenting with such real data cannot provide the other aspects of the prediction systems. Furthermore, synthetic data can be generated in different ways to see how the models perform with different ratios and distributions. During model building and prediction, it is assumed that the data set is cleaned.

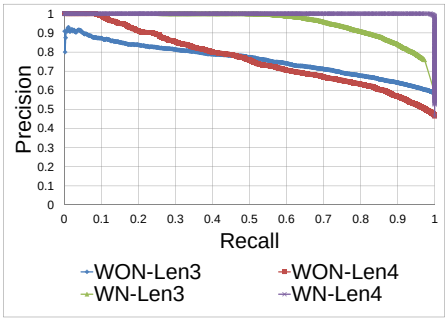
Synthetic data sets. We generate 5 different data sets for the airport baggage tracking scenario, where bags follow the paths shown in the floor plan in Fig 4.1. There are two preplanned paths, *P1*: $l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_5 \rightarrow l_7$, and *P2*: $l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_6 \rightarrow l_8$. There are 20 flights a day and each flight departs after every 30 to 60 minutes. The first flight of a day starts at 8:00 am. In each of the data sets, there are a total of 5K flights carrying 100K bags. Each of the flights has 20 registered bags. Each data set contains approximately 450K stay records. Flight IDs and bag IDs are generated sequentially and the flight with even IDs are allocated to path P1 and others to P2. Bags are checked in at the earliest 3 hours and the latest 30 minutes before the flights. For each possible transition, bags follow a realistic range of duration with different distributions that will be discussed next. In our example scenario, the transitions $l_1 \rightarrow l_2$ and $l_2 \rightarrow l_3$ have less influence on baggage mishandling, whereas the sorters (i.e., $l_3 \rightarrow l_5$, $l_3 \rightarrow l_4$, and $l_4 \rightarrow l_6$) have higher influence. So, we put relatively smaller time intervals for those transitions. The duration ranges (in



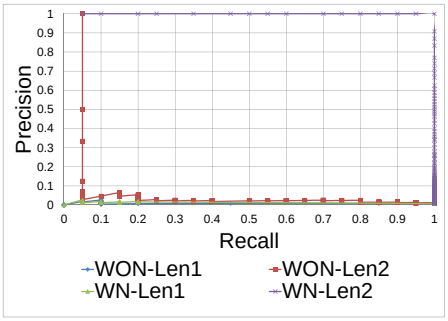
(a) P1-PFG vs. P1-APFG for DS_4



(b) P_{r2}-PFG vs. P_{r2}-APFG, DS_R



(c) With vs. without norm., DS_1



(d) With vs. without norm., DS_R

Fig. 4.7: PR curves for comparing PFG vs. APFG, and with (WN) vs. without normalization (WON) while using APFG

seconds) for different paths are, P1: $l_1 \xrightarrow{25-60} l_2 \xrightarrow{50-300} l_3 \xrightarrow{120-9000} l_5 \xrightarrow{400-500} l_7$, and for P2: $l_1 \xrightarrow{25-60} l_2 \xrightarrow{50-300} l_3 \xrightarrow{120-2500} l_4 \xrightarrow{120-5400} l_6 \xrightarrow{400-500} l_8$.

Varying the distributions of the durations for the transitions and durations before the flights, we generate 5 different data sets (DS). Each data set is divided into a training set (TRS) and a test set (TSS) containing 70K and 30K bags, respectively. It is also made sure that the bags for the same flight are not be distributed between training and test set as it might give a biased estimation due to overfitting. We also use validation sets from the TRS for cross validation while finding the best value for RS_{th} that will be discussed later in this section. The data sets are described below:

- **DS_1:** Transition durations and durations before flights are uniformly distributed. DS_1 contains 53% mishandled bags.
- **DS_2:** Transition durations follow a normal distribution and durations before flights follow uniform distribution. DS_2 can show the effect in the models when the transition durations are normally distributed com-

pared to the uniform distribution of DS_1. DS_2 contains 54% mishandled bags.

- **DS_3:** Transition durations follow a log-normal distribution and durations before flights follow uniform distribution. As a log-normal distribution creates long tail, it generates less mishandled bags compared to DS_1 and DS_2. This distribution reflects a more realistic scenario of airport baggage tracking. The data set contains 12% mishandled bags.

- **DS_4:** Transition durations follow a log-normal distribution with different μ and σ compared to DS_3 and durations before flights follow a normal distribution. The main intention is to reduce the mishandling rate to below 2%. It also can expose how good the models are when the mishandling rate is very low. DS_4 contains 1.42% mishandled bags.

- **DS_5:** The distributions of durations are similar to DS_4. However, the bags are flowed from the opposite direction. So, in DS_5, $P1{=}l_7 \rightarrow l_5 \rightarrow l_3 \rightarrow l_2 \rightarrow l_1$, and $P2{=}l_8{\rightarrow}l_6{\rightarrow}l_4{\rightarrow}l_3{\rightarrow}l_2{\rightarrow}l_1$. As seen, the change in direction of the path brings the sorter in the earlier step. In the sorter bags generally spend most of its operational time and considered as the bottleneck of the system. The data set can show how bringing bottleneck earlier in the path can affect the models. DS_5 contains 1.53% mishandled bags.

Real data sets (DS_R). We use a small real RFID baggage tracking data from an airport $A1$. For the reason of confidentiality, the airports' names are not disclosed. The bags are originated from $A1$ to the destination airport $A2$. There are 6 RFID readers deployed. Two of them for arrival system and 4 of them are for departure system. Our application scenario is interested only in the departure system. From the data set we derived three different preplanned paths, P_{r1}: Check-in→Sorter→GateWay-1→BeltLoader, P_{r2}: Check-in→Sorter→GateWay-1, and P_{r3}: Check-in→Sorter→BeltLoader. After removing many noisy records, we have a total of 20.4K bags for 2.5K different flights. There are only 75 mishandled (MH) bags which are only 0.35% of the total bags. The details for the training set are: total 15.9K, MH 29 (0.18%), P_{r1}-[total 1.5K, MH 7], P_{r2}-[total 10.1K, MH 4], P_{r3}-[4.3K, MH 18]. The test set details are: total 4.6K, MH 43, P_{r1}-[total 1.5K, MH 20], P_{r2}-[total 2.6K, MH 20], P_{r3}-[total 0.5K, MH 3].

6.2 Test cases

We build PFGs and APFGs from all the mentioned data sets and tested them from various perspectives. The PFGs and APFGs are built from the combined records (i.e., containing all the paths in the data sets) called C-PFG,

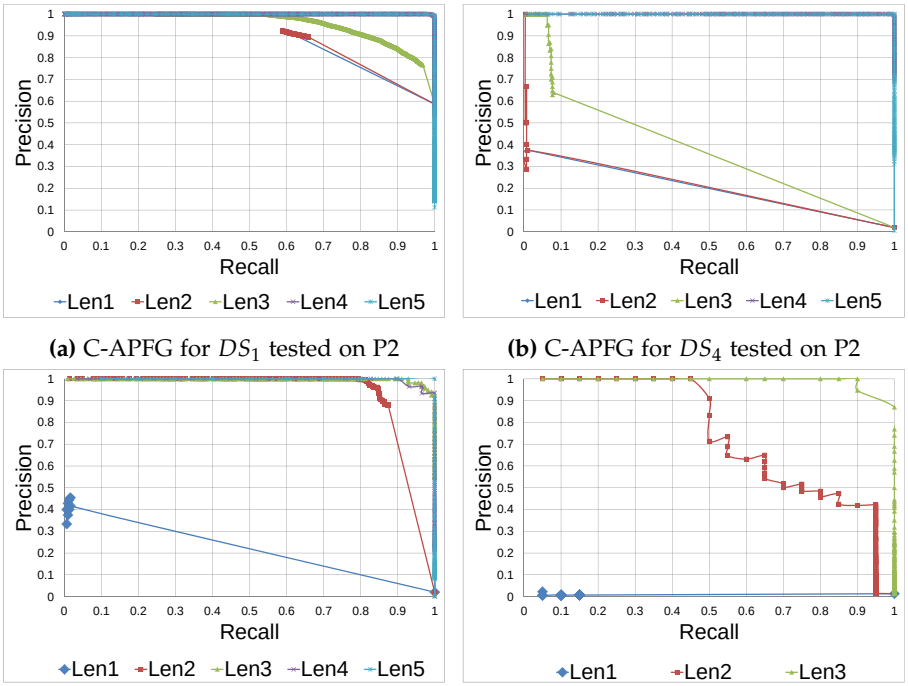
and C-APFG, respectively. We also separately build PFGs and APFGs with the records of each different path, e.g., P1-PFG, P1-APFG for P1 in synthetic data, P_{r1}-PFG, P_{r1}-APFG for P_{r1} in real data, etc. The PFGs and APFGs are tested on the test set for the relevant paths. We test them both without normalizing (WON) and with normalizing (WN) the durations before flights.

We apply the PFGs and APFGs on all the bags of the TSS and for each bag, we obtain a risk score for each of its transitions based on their transition duration. For each of the generated risk scores r, we compute the *recall*, where *recall(r)*= $\frac{\text{\# of mishandled bags having risk score} \geq \text{r}}{\text{\#of mishandled bags}}$. Conversely, for each r, we also compute the *precision*, where *precision(r)*= $\frac{\text{\# of mishandled bags having a risk score} \geq \text{r}}{\text{\#of bags having a risk score} \geq \text{r}}$. In our scenario, a perfect precision score of 1 means that all the classified mishandled bags are also actually mishandled. However, this precision score says nothing about whether all mishandled bags are predicted correctly. Conversely, a perfect recall score of 1 means that all the actually mishandled bags are classified as mishandled. However, this recall says nothing about how many correctly handled bags are wrongly predicted as mishandled. We draw *precision-recall (PR) curves* that represent how precision and recall changes with the risk scores. The perfect point in a PR curve is (1,1) that represents the predictions for the mishandled bags are perfect and any correctly handled bags are not predicted as mishandled.

We generate PR curves for the various test cases discussed above and report only the cases that are interesting to analyze. For all the test cases we analyze the PR curves for the different transition lengths, e.g., for path P1 there are 4 different lengths of transitions l_1 to l_2 (*Len1*), l_1 to l_3 (*Len2*), ..., and l_1 to l_7 (*Len4*). Similarly, P2 has 5 different lengths of transitions. For the real data, P_{r1} has 3 different transitions, P_{r2} and P_{r3} have 2 different lengths of transitions. The PR curves for the different test cases are reported in Fig. 4.7 and 4.8. The PR curves are analyzed from the different perspectives and explained next. The PR curves for DS_2 and DS_3 are not reported as they show the same behavior as others.

6.3 Analyzing the PR Curves

PFGs vs. APFGs. The PR curves for comparing the PFGs and APFGs are shown in Fig. 4.7a and 4.7b. Fig. 4.7a reports the PR curves generated by applying P1-PFG and P1-APFG on the TSS for P1 in DS_4. It reports the results for Len3 and Len4. PFG-Len3 can be compared with APFG-Len3 and so on. The results show that the APFGs always provide higher precisions compared to the PFGs. As we found same type behavior for the other DSs, they are not reported. Fig. 4.7b shows the similar experiments with DS_R. For Len1 PFG and APFG provide the same score. So, the lines are overlapped. However, for Len2, the APFG provides relatively better results. Overall from the experimental results, it is clear that the APFGs can better capture the riskiness and



(a) C-APFG for DS_1 tested on P2

(b) C-APFG for DS_4 tested on P2

(c) C-APFG for DS_5 tested on P2

(d) C-APFG, DS_R tested on P_{r1}

Fig. 4.8: Comparing the effect of path lengths and location types on APFGs while applying with normalization

recovery of the objects and better differentiate between the correctly and incorrectly handled bags. In rest of the experiments, we report only the results with APFG.

With and without normalization. The PR curves for comparing the results with and without normalizing the duration before flights are presented in Fig. 4.7c and 4.7d. In Fig. 4.7c, P2-APFG is applied on the records for P2 in DS_1. In Fig. 4.7d, we use the same APFG used for Fig. 4.7b. In all cases, the results show that the normalizing boosts the performance. In all the cases except WN-Len1 (Fig. 4.7d), we can get almost a perfect classification, i.e., close to full precision with 100% recall when normalizing. The results for WN-Len1 can be understood better in the next paragraph.

Influence of path length and location type. Fig. 4.8 reports the PR curves for showing the effect of path length on the classification performance. Overall, the results show that the performance gets better with increasing the path length. In the case of DS_1 (Fig. 4.8a) and DS_4 (Fig. 4.8b), Len1 and Len2 have less influence on a baggage mishandling. These transitions also take very short durations. As a result, the performance is poor up to those transitions.

However, in the case of DS_5, which contains the same distribution as DS_4, with the direction of the path is reversed, the performances from Len2 and afterward are close to perfect classification. The main mishandling occurs in the sorting system as a bag takes longer for completing its sortation. So, it shows that the model can classify mishandled bags very accurately when they come in the bottleneck in their path and the result continues getting better as objects move forward. The result with DS_R also shows the similar behavior (Fig. 4.8d).

Combined model vs. specialized model for each path. Comparing the results of Fig. 4.7c with Fig. 4.8a shows that testing the bags with path P2 by the C-APFGs and P2-APFGs provides the same result. All the cases in our experiments we found that testing the bags with combined model and specialized model provide the same results. However, it is also true that the ALDPHs of a C-APFG become specialized for the different paths when they start following different path sequences. In our generated data set, P1 and P2 have a common path from l_1 to l_3. After that, they follow different path sequences. So, it will be best to use only combined model, instead of building many models for different paths.

Effect of data distribution and mishandle ratio. In general, the data distributions do not change the overall behavior of the models. In DS_1, the mishandling rate is balanced. So, it starts giving very good precision from Len1 (Fig. 4.8a) compared to the other cases where the mishandling rate is extremely low (Fig. 4.8b to 4.8d). However, the models built from all our different data set provide very good results. It also shows the proposed APFG with normalization can perform very well in an imbalanced class situation and does not get affected by the class imbalance problem.

6.4 Finding the best RS_{th}

We use DS_4 in this experiment. TRS of DS_4 is divided into 4 folds for the k-fold cross validation. Each fold contains 17.4K bags, where almost half of them belong to P1. It is also made sure that the bags from the same flight are not distributed to multiple folds. Iteratively, we learn APFGs from 3 folds and use the 4th fold for testing. So, finally we have the test results for 4 APFGs. We use Eq. (4.7) with x=\$96 (according to [13]) and y=\$15 (salary of a baggage handler is app. \$13/hour) for obtaining the benefit for the different RSs in the results. For optimizing the RS_{th} for each path length, we take the average of the risk scores that provide the highest benefit in each of the 4 models. Then the average RS is used as the RS_{th} for predicting the riskiness of the actual test bags. The test results are analyzed from two different perspectives. In scenario1 (SC1), the predicted mishandled bags are not removed from the system unless they automatically disappear when they are really mishandled. It can show the actual benefit at different path lengths. In scenario2 (SC2),

Table 4.5: Results based on the selected RS_{th}

Pathlen	RS_{th}	SBF	Bnft-SC1	Bnft-SC2	pred->	P-SC1	N-SC1	P-SC2	N-SC2
Len1	1	30.6	168	168	Act-P	3	283	3	283
		min			Act-N	5	14709	5	14709
Len2	0.89	27.8	102	0	Act-P	2	283	0	283
		min			Act-N	4	14710	0	14709
Len3	0.08	26.9	1587	1419	Act-P	22	264	19	264
		min			Act-N	13	14701	8	14701
Len4	0.46	18.2	22701	21324	Act-P	281	0	264	0
		min			Act-N	4	14710	4	14697
Len5	0.48	10.3	5265	0	Act-P	65	0	0	0
		min			Act-N	0	14714	0	14697

at different path lengths, the predicted mishandled bags are removed from the system such that they cannot be seen in the subsequent locations in their path. It can show the total benefit if bags are saved whenever it is detected as mishandled. The selected RS_{th}s, benefits, confusion matrices, and how early the bags are saved before the flight (SBF) for the bags with preplanned path P2 are reported in Table 4.5. As there are 286 mishandled bags, the total cost without using the ORP will be \$27456. The benefits with SC1 shows that handling bags only at L_4 can save 82.7% of the total mishandling cost. Whereas, in SC2, $\frac{168+0+1419+21324+0}{27456} \times 100 = 83.4\%$ of the total cost can be saved. It also shows that all the mishandled bags are predicted within Len4 and at least 18.2 minutes before the flights.

6.5 Scalability

We use DS_4 for building PFGs and APFGs for showing the scalability regarding their construction time and memory use. We use a set of SQL queries with some DDL and DML operations for building PFGs and APFGs and they are stored in the database tables. The SQLs queries for PFG and APFG are shown in listing 4.1 and řefAPFGConstruction, respectively. Before prediction, a C# program loads the PFG and APFG into main memory. The full PFG and APFG construction and loading times are reported in Fig. 4.9a. In the case of SQL query times, we clear the cache after executing each operation. In all cases, we run the queries and code 3 times and report the rounded average time. In both cases, the results show that the construction time increases linearly with the number of bags. We also report the memory use of LDPHs and ALDPHs for the different numbers of bags. It shows that the size grows linearly with the number of bags. It also shows that the total size of ALDPHs is on average 84% higher than the total size of LDPHs. However, the total size of the ALDPHs is very small, only 194 KB for 70K bags. So, it is feasible even for a larger data set.

Listing 4.1: PFG Construction

```
SELECT FromLocation, ToLocation, Duration, Count(*)
   DurationCount
INTO PFGInfo FROM StayRecord WHERE BagId<=70000
GROUP BY FromLocation, ToLocation, Duration
ORDER BY FromLocation, ToLocation, Duration;

ALTER TABLE PFGInfo ADD  TransitionCount int,
   CountDurGt int, LDP float;

UPDATE PFGInfo SET TransitionCount = (SELECT SUM(
   DurationCount) FROM PFGInfo T1
WHERE T1.FromLocation=PFGInfo.FromLocation AND T1.
   ToLocation=PFGInfo.ToLocation);

UPDATE PFGInfo SET CountDurGt = (SELECT SUM(
   DurationCount) from PFGInfo T1
WHERE T1.FromLocation=PFGInfo.FromLocation and T1.
   ToLocation=PFGInfo.ToLocation AND
T1.duration>=PFGInfo.duration);

UPDATE PFGInfo set LDP = ROUND(CountDurGt*1.0/
   TransitionCount, 4);
```

Listing 4.2: APFG Construction

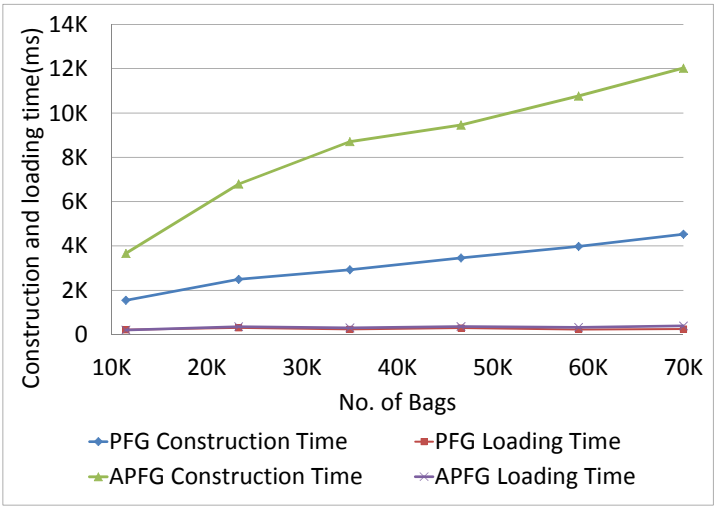
```
SELECT BagID, PathString, TotalDuration,
   CompleteStatus, PathLength, FinalStatus INTO
   APFGInfo
From StayRecord  WHERE BagID<=70000  ORDER BY BagID;

SELECT distinct PathString, TotalDuration, COUNT(*)
   CountPathDur INTO ALDPHInfo
FROM  APFGInfo WHERE PathLength>1 GROUP BY  PathString
   , TotalDuration  ORDER BY PathString, TotalDuration

ALTER TABLE ALDPHInfo ADD COUNTPath int,
   CountPathDurGt int, ALDP float;

UPDATE ALDPHInfo SET COUNTPath = (SELECT SUM(
   CountPathDur) FROM ALDPHInfo T1 WHERE T1.PathString
    = ALDPHInfo.PathString);

UPDATE ALDPHInfo SET CountPathDurGt =
(SELECT SUM(CountPathDur) FROM ALDPHInfo T1 WHERE T1.
   PathString = ALDPHInfo.PathString AND T1.
   TotalDuration>=ALDPHInfo.TotalDuration);

 UPDATE ALDPHInfo SET ALDP = round(CountPathDurGt*1.0/
    CountPath, 4);
```

6.6 General lessons

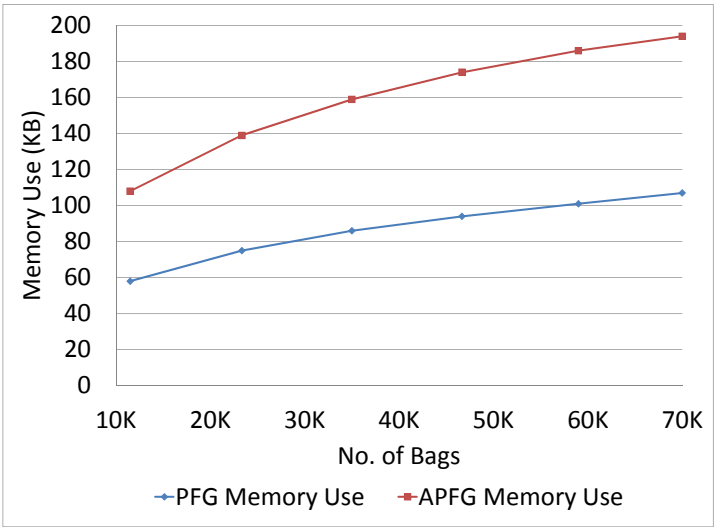
From the experimental results, we can say that APFG produces the best result and the available processing duration must be normalized before producing a risk score of an object. The prediction accuracy for the mishandled objects boost up when they reach the bottleneck locations in their paths. The prediction system behaves in a similar way in a class imbalanced data sets. The prediction system can save the cost for mishandling significantly. The proposed APFG also scalable to large data set.

7 Related Work

Related work falls into two main categories. One is to pre-process raw indoor tracking data and another one is to perform data mining on such tracking



(a) Construction Time



(b) Memory Usage

Fig. 4.9: PFG and APFG construction times and memory usage.

data. The RFID data management challenges and some solutions are discussed in [27, 76]. Data warehousing, mining, and workflow analysis are proposed for RFID-based item tracking in the supply chain systems in [36, 37]. The authors convert the raw RFID records into cleansed record containing the first and last reading times of an object under the readers activation range.

In the present chapter, we use stay records discussed in [15], as it can capture the total stay duration between locations. Graph-based model for indoor tracking is discussed in [16, 46]. In the present chapter, we extend graph models for capturing the object flows with new probabilistic concepts such as LDP, ALDP, and histograms.

Data mining is performed on the tracking data for finding frequent spatio-temporal sequential patterns [26, 44], typical movements of objects in indoor space [69], frequent trajectory patterns for activity monitoring [57], and frequent walk in RFID-equipped warehouse [23]. Interesting spatio-temporal rule mining applications, techniques and issues are discussed in [35]. The present chapter introduces a new perspective which is for risk prediction in indoor moving object. In [13], RFID baggage tracking data are analyzed for mining risk factors in the offline scenario. The present chapter focuses on an online risk prediction scenario that require more fine grained features such as object transitions at the reader level and duration for each of the transitions. Further, this chapter is more general as the used features are common in many symbolic indoor and mixed indoor-outdoor tracking applications.

8 Conclusion and Future work

We proposed detailed steps and probabilistic models for predicting the risk of online indoor moving objects. We converted the historical raw tracking records into stay records and use them for constructing the probabilistic flow graphs called PFG and APFG. The graphs capture the probabilistic information about the transition times by using histograms called least duration probability histogram LDPH and aggregated LDPH (ALDPH). The flow graphs are used for obtaining risk score of an online indoor moving object and for predicting risks. A comprehensive experiment with synthetic and real data showed that the proposed risk prediction method can differentiate risky objects from the correctly handled objects very accurately when the objects approach the bottleneck locations on their paths. We also proposed a cost model for object mishandling and the experiments showed that using APFG with the proposed normalization can significantly save the operation cost. The result also showed that the risky objects are predicted early enough such that they can be saved from being mishandled.

In future work, the proposed techniques can be expanded to more general scenarios such as mixed indoor-outdoor object tracking. Further, predicting risks for the objects in nondeterministic scenarios, where the paths of the objects are unknown in advance, can be another future direction.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

Chapter 5

Finding Dense Locations in Symbolic Indoor Tracking Data: Modeling, Indexing, and Processing

The paper is under revision for publishing in a Journal. This paper is an extended version of our two previous conference papers [14, 16], which are presented in Appendix A and B, respectively.

Abstract

Finding the dense locations in large indoor spaces is very useful for many applications such as overloaded area detection, security control, crowd management, indoor navigation, and so on. Indoor tracking data can be enormous and are not immediately ready for finding dense locations. This chapter presents two graph-based models for constrained and semi-constrained indoor movement, respectively, and then uses the models to map raw tracking records into mapping records that represent object entry and exit times in particular locations. Subsequently, an efficient indexing structure called Dense Location Time Index (DLT-Index) is proposed for indexing the time intervals of the mapping table, along with index construction, query processing, and pruning techniques. The DLT-Index supports very efficient aggregate point, interval, and duration queries as well as dense location queries. A comprehensive experimental study with both real and synthetic data shows that the proposed techniques are efficient and scalable and outperforms RDBMSs significantly.

1 Introduction

Technologies like radio frequency identification (RFID) and Bluetooth enable a variety of indoor tracking applications such as tracking people's movement in large indoor spaces (e.g., airports, office buildings, shopping malls, and museums), airport baggage tracking, item movement tracking in supply chain systems, etc. The massive amount of tracking data generated by such systems is very useful for analyses and decision-making in indoor application scenarios. These analyses are useful for different kinds of location-based services, finding problems in the systems, and further improvement in the systems. Unlike GPS-based positioning for outdoor applications, indoor tracking in our research provides the symbolic locations of the objects in indoor space. Examples of symbolic locations are security and shopping areas in airports, different sections or rooms in museum exhibitions, etc. In airport baggage handling, the bags pass different symbolic locations such as check-in, screening, sorting, etc. Detecting dense locations in a baggage handling scenario can identify overloaded baggage handling locations, which in turn can contribute to better baggage handling. By analyzing tracking data on passengers in an airport, we can find out where and when passengers gather, and such information can be used for crowd management in the airport.

It has been pointed out that the geometric polyline representation for outdoor trajectories is unsuitable for indoor trajectories 47. For example, consider an RFID tracking application where RFID readers are deployed at the doors of different rooms in a large indoor space. Each reader has a very limited tracking range that covers a small portion of the room, e.g., only the door of a room. If an object containing an RFID tag moves from one room to another, this produces two consecutive tracking records of the object location in different rooms. Due to limitations in indoor positioning technologies, the locations of the object between these two records are not obtained. Moreover, indoor tracking systems like airport baggage and people movement generate large volumes of data, making efficient query processing techniques essential. Taking all of these challenges into account, this chapter proposes an efficient approach for extracting dense locations from indoor tracking data.

The chapter has several contributions. First, two graph-based models are proposed for capturing movements of objects inside symbolic indoor spaces, one for constrained movements and another for semi-constrained movements. Second, two kinds of mappings are designed, one for constrained and another for semi-constrained movements. The mappings are used to map the indoor tracking records into mapping records capturing the entry and exit times of the objects at different symbolic indoor locations. The resulting mapping records can be used for dense location extraction as well as for many other analyses, e.g., stay duration estimation. Third, two

different types of densities, namely, *count based density* (CBD) and *duration based density* (DBD), are defined for different application scenarios. Fourth, an efficient indexing structure, the Dense Location Time Index (*DLT*-Index), is proposed. It stores aggregate information, i.e., the number of (unique) objects entering, exiting, and present at a location at different timestamps or time intervals. Fifth, efficient data processing techniques are proposed, including algorithms for *DLT*-Index construction and updating, data pruning methods in the *DLT*-Index, and algorithms for finding dense locations using the *DLT*-Index. Efficient processing algorithms for aggregate *point, interval*, and *duration queries* via the *DLT*-Index are also proposed. The developed *DLT*-Index is generalized such that it can be used for indexing any types of time intervals and it enables us to query for the distinct number of records at a given time point, as well as for a given time interval. Sixth, a comprehensive experimental evaluation using both real and synthetic data is conducted, and the results show that the proposed solution is efficient and scalable.

This chapter is an extended version of our two previous conference papers [14, 16], which are presented in Appendix A and B, respectively. Most importantly, it adds a new density definition (Section 2.2), related query processing techniques for the new density definition (Section 5.3), and an experimental evaluation (Section 6.4) of the proposed query processing techniques. In addition, the chapter adds algorithms for data mapping in constrained and semi-constrained path spaces (Section 3), algorithms for *DLT*-Index construction (Section 4.3), and query processing algorithms for other query types using the *DLT*-Index (Section 5). Further, the chapter also provides a formal definition of the proposed graph for semi-constrained path spaces (Section 3.3), update strategy of the *DLT*-Tree (Section 4.4), and pertinent experiments with both synthetic and real data (Section 6).

The remainder of the chapter is organized as follows. Section 2 gives the problem formulation. Section 3 describes the graph-based models and the mapping of tracking records. Section 4 presents the general steps of processing a dense location query and describes the *DLT*-Index. Section 5 presents the query processing and pruning techniques over the *DLT*-Index. Section 6 presents the experimental evaluation. Section 7 reviews related work. Finally, Section 8 concludes the chapter and points to future research.

2 Problem Formulation

2.1 Problem Scenario

The movements of objects inside indoor spaces vary with the structure of the paths. Based on the path structure, we categorize the indoor spaces into two categories.

Constrained Path Space (CPS): In a constrained path space (CPS), objects move continuously from one symbolic location to another. The objects cannot move freely inside the locations, and the locations are in some sense one-dimensional. The size of the locations inside a CPS is measured by length, not by area.

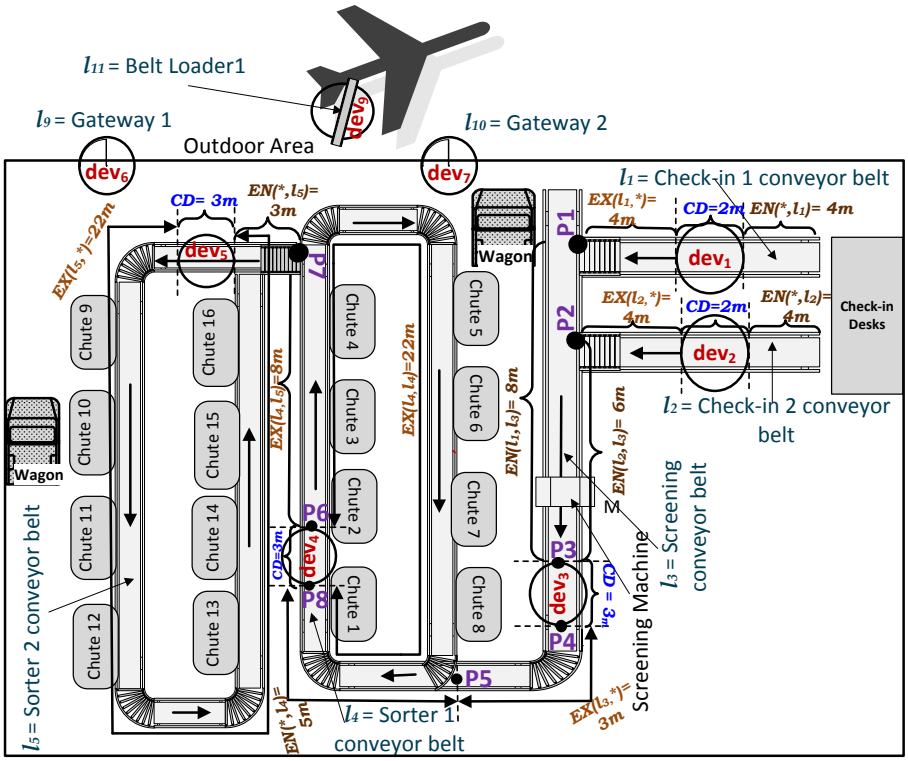


Fig. 5.1: Constrained path space

Fig. 5.1 shows an example of a CPS, which is a conveyor system of an airport baggage handling system. The conveyor system is divided into different symbolic locations like check-in 1, check-in 2, screening, sorter-1, and sorter-2.

Semi-Constrained Path Space (SCPS): In a semi-constrained path space (SCPS), the objects move more freely than in a CPS. The objects move from one symbolic location to another, and they can also stay some period of time inside the locations. The locations are two-dimensional. The size of the locations inside a SCPS is measured by area, not by length.

Examples of the SCPSs include office spaces, museums, different sections in airports, etc. Fig. 5.2 shows an example of SCPS. The indoor space is divided into different symbolic locations, e.g., rooms, hallways, etc. An object

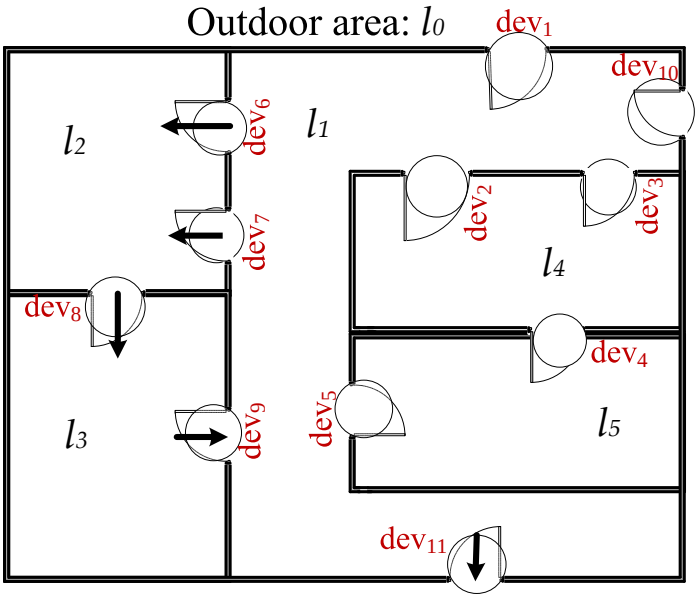


Fig. 5.2: Semi-constrained path space

has to cross the entry/exit points (e.g., doors) for entering and exiting a room. Some entry/exit points are uni-directional, and some are bi-directional. The arrows represent the uni-directional movements. Moreover, a large hallway or corridor (e.g., l_1 of Fig. 5.2, long corridor in an airport, etc.,) can be divided into multiple sections where each section is considered as a separate location.

In our setting, the tracking devices are strategically deployed at different fixed locations inside the indoor space, e.g., at a specific point of each section of the conveyor belts, by the doors of a room, between sections of a hallway, etc. Specifically, a location inside the CPS contains one tracking device deployed at any point in the location, whereas for the SCPS a tracking device is deployed at the entry and exit point of each location. The objects hold tags or devices that can be tracked by the tracking devices. For example, in the case of RFID technology, objects with RFID tags are tracked by RFID readers. For the Bluetooth systems, Bluetooth access points track objects with Bluetooth enabled devices. After deployment of the tracking devices, the positions are recorded in the database. In Fig. 5.1 and 5.2 the circles represent the deployment of the RFID readers and their activation ranges. When an object enters into the activation range of a tracking device, it is continuously detected by the tracking device with a given sampling rate and it generates raw reading records in the form: *(TrackingDeviceID, ObjectID, t)*. This means that a tracking device *TrackingDeviceID* detects a moving object *ObjectID* in its activation

range at timestamp t. A *TrackingRecord(recordID, ObjectID, TrackingDeviceID, $time_{in}$, $time_{out}$)* table is constructed from the raw tracking sequences, where *recordID* is a record identifier and $time_{in}$ *and* $time_{out}$ represent the timestamps of the first and last readings of *ObjectID* by *TrackingDeviceID* in its activation range, respectively. Table 5.1 shows a set of example tracking records of an object o_1 in the floor plan of Fig. 5.1. Here the record rec_1 means that object o_1 is observed by tracking device dev_1 from time 4 to 5, and record rec_3 means that o_1 is observed by dev_3 from time 15 to 18. Due to the limitation of indoor positioning systems, it is unknown at what position of o_1 is between 6 and 14 without knowing the floor plan.

Table 5.1: Tracking Records of Indoor Moving Objects

RecordID	ObjectID	TrackingDeviceID	$time_{in}$	$time_{out}$
rec_1	o_1	dev_1	4	5
rec_3	o_1	dev_3	15	18
rec_5	o_1	dev_4	26	29
rec_8	o_1	dev_4	51	54

2.2 Problem Definition

Let L be the set of all symbolic locations inside a large indoor space, $L = \{l_1, l_2, l_3, \ldots, l_k\}$. We assume a discrete equi-distant time domain composed of *time points* represented by integers. Depending on the requirements of the application scenario, we pick the *time unit* (the granularity of the time points) to be, e.g., 1, 5, or 15 minutes. A time interval is specified by the start and end time points.

Definition 1 (Capacity). The *capacity* of a location $l_i \in L$, denoted as *capacity*(l_i), is the number of objects that can be present in l_i during a time unit.

For example, in Fig. 5.2, the time unit is 15 minutes and the *capacity* of room l_3 is 20 persons per time unit (i.e., 20 persons per 15 minutes). Similarly, in Fig. 5.1, the time unit is 1 minute and the capacity of *check-in 1* conveyor is 20 objects per time unit (i.e., 20 objects per minute).

Definition 2 (Count Based Density (CBD)). Let n_i be the number of objects present in location l_i during the time interval $w = [t_{start}, t_{end}]$ and capacity(l_i) be the capacity of location l_i. The *count based density (CBD)* of location l_i for interval w is defined as,

$$CBD_i = \frac{n_i}{\Delta t \times capacity(l_i)}, where \;\; \Delta t = t_{end} - t_{start} \tag{5.1}$$

Definition 3 (Duration Based Density (DBD)). Let, $O_i = \{obj_1, obj_2, obj_3, \ldots$

, obj_n} be the set of objects present in location l_i during the time interval w = [t_{start}, t_{end}], and dur_{obj_i} be the time spent by object $obj_i \in O_i$ at location l_i within the time interval w. So, the total length of the stays of $n_i = |O_i|$ objects in location l_i during the time interval w is: $dur_i = \sum_{i=1}^{n} dur_{obj_i}$. Let, capacity($l_i$) be the capacity of location l_i. The *duration based density (DBD)* of location l_i for interval w is defined as,

$$DBD_i = \frac{dur_i}{\Delta t \times capacity(l_i)}, where \;\; \Delta t = t_{end} - t_{start} \tag{5.2}$$

As seen, for both densities, the density is computed by dividing the actual number of objects (for CBD) or their stay duration (for DBD) by the location's capacity. In both cases, this results in values normalized in the range $[0, 1]$.

An example scenario of capacity and computation of density is illustrated in Fig. 5.3. In the figure, each empty box represents a free slot of the location at the given time point. For example, at time t_1, two slots are free. At time t_3, there are no empty slots. At t_3, o_1 exits the location and two objects o_3 and o_4 enter the location. During the query time interval $[t_1, t_3]$, a total of four objects {o_1, o_2, o_3, o_4} present in the location, and the total length of the stays by all the objects is 6. As seen, CBD considers only the number of objects and not the stay duration of the objects, whereas DBD does. However, if the average time spent by the objects at a location is one time unit, CBD is equal to DBD. CBD is more appropriate for the scenario of the conveyor belts system where objects move continuously with a constant speed. Whereas, DBD is more suitable for the people movements in a semi-constrained indoor scenario where the stay of a person at a location varies. As the duration part is not considered in CBD, it always holds that CBD $\leq$ DBD.

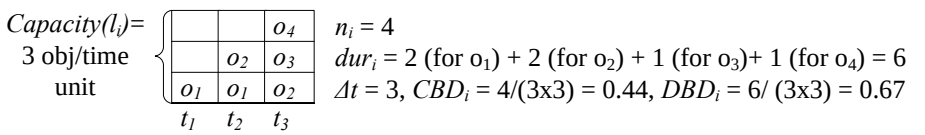


Fig. 5.3: Example of density computation

Definition 4 (Count Based Dense Location (CBDL)). Given a threshold θ, a location l_i is considered as a *count based dense location (CBDL)* for interval w if its $CBD \geq \theta$.

Definition 5 (Duration Based Dense Location (DBDL)). Given a threshold θ, a location l_i is considered as a *duration based dense location (DBDL)* for interval w if its $DBD \geq \theta$.

Definition 6 (Count Based Dense Location Query (CBDLQ)). A *count based dense location query (CBDLQ)* finds all the *CBDLs* $\subseteq L$ for a given time interval w.

Definition 7 (Duration Based Dense Location Query (DBDLQ)). A *duration based dense location query (DBDLQ)* finds all the *DBDLs* $\subseteq L$ for a given time interval w.

For example, consider Fig. 5.3. Let $\theta = 0.65$. As a result, the location is not a CBDL for the time interval $w = [t_1, t_3]$, because its $CBD = 0.44 \ngeq \theta = 0.65$. Whereas, the location is a DBDL for the time interval w, because its $DBD = 0.67 \geq \theta = 0.65$.

3 Semantic Location Mappings

A location is typically not fully covered by a tracking device. Moreover a tracking record contains only the first and last times an object appeared inside the activation range of a tracking device. As a result, it is explicitly unknown when an object actually entered and exited the corresponding location. To this end, mapping strategies are required to retrieve such location and timing information. We create a *mapping table* with *mapping records* that are derived from tracking records, where a mapping record is in the form $\langle$ *MappingID, ObjectID, LocationID,* $time_{start}$, $time_{end}$ $\rangle$. A mapping record represents that an object *ObjectID* entered location *LocationID* at time $time_{start}$ and exited at time $time_{end}$.

3.1 Modeling Symbolic Locations

As discussed earlier the deployment of tracking devices varies between CPS and SCPS locations. The model for CPS is more detailed as the tracking device can be deployed at any point inside a location. For example, in Fig. 5.1 *check-in 1* is represented by dev_1. When a bag goes to *sorter-1* it will be read by dev_4 and then it may go to *sorter-2* or *chute* or it may circulate within *sorter-1*. On the contrary, a location inside the SCPS may have many entry and exit points where tracking devices are deployed. Moreover, the movement can be both uni-directional and bi-directional. For example, in Fig. 5.2 the door with dev_1 can be used for both entering and exiting the location l_1. However, the door with dev_6 can be used only to enter l_2.

3.2 Modeling CPS

In order to map the tracking records with the semantic locations and the entry and exit times of objects, an Extended Reader Deployment Graph (ERDG) is constructed from the indoor plan (e.g., given in Fig. 5.1). Some definitions are needed to understand the ERDG:

Definition 8 (Covered distance). Given a path p and a tracking device d, the covered distance (CD) is the length of the part of p that is covered by d's

detection range. The CD for a tracking device dev_1 is denoted as *CD(dev$_i$)*. For example, in Fig. 5.1 *CD(dev$_1$)* = $2m$ shows the CD of dev_1 at l_1.

Definition 9 (Entry lag distance). The *entry lag distance (ENLD)* from location l_x to l_y denoted as $EN(l_x, l_y)$ is the distance from the ending point of l_x to the first reading point at l_y.

For example, consider Fig. 5.1. The journey of an object o at location l_3 can start from either points $P1$ or $P2$ depending on whether o is coming from l_1 or l_2. While moving through l_3, o will be first tracked by dev_3 when it arrives at point $P3$. Here, the distance between the point $P1$ and $P3$ is an *entry lag distance (ENLD)* which is denoted as $EN(l_1, l_3)$. A location l_y can have many ENLDs depending how many locations end at l_y. In our example $EN(l_1, l_3) = 8m$ and $EN(l_2, l_3) = 6m$. A special notation $EN(*, l_y)$ is used, which indicates that the ENLD at l_y is the same, regardless of where an object is coming from. In our example, $EN(*, l_4) = 5m$ is the ENLD of location l_4 from any location ended at l_4.

Definition 10 (Exit lag distance). The *exit lag distance (EXLD)* from location l_x to l_y denoted as $EX(l_x, l_y)$ is the distance from the last reading point at l_x to the exit point of l_x that leads to location l_y.

In Fig. 5.1, the journey of an object at location l_3 ends when it passes the point $P5$ and reaches location l_4. While traveling through l_3 the object was last detected by dev_3 when it was at point $P4$. Here, the distance between P4 and P5 is the *exit lag distance (EXLD)* of l_3 which is denoted as $EX(l_3, l_4)$. As l_3 has only one destination, the EXLD of l_3 is always the same, regardless of the destination. So in this case, $EX(l_3, *)$ is used instead of $EX(l_3, l_4)$. In our example, $EX(l_3, *) = 3m$. Similar to ENLD, a location can have many EXLDs. For example, an object can leave location l_4 by going to l_5 through point $P7$ or can circulate in l_4 and leave through any point between $P6$ and $P8$. As a result, l_4 has two EXLDs: $EX(l_4, l_5)$ = $8m$ and $EX(l_4, l_4)$ = $22m$.

The ERDG is formally defined by a labeled directed graph G = *(L, E, T, lb$_E$)*, where:

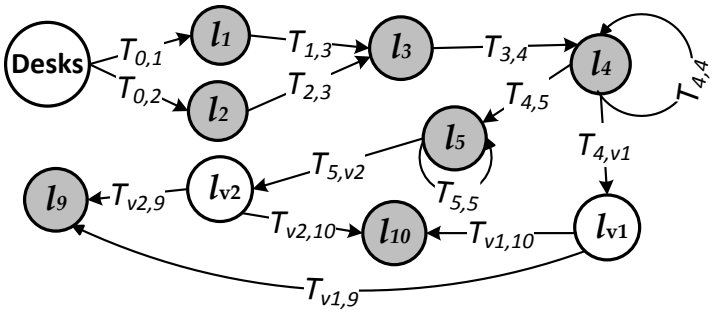
1. L is the set of locations where each location is represented as a vertex in G. If a location does not have any tracking device deployed in it, the corresponding location is labeled as a *virtual location* l_{v_x}, where x is an integer.

2. E is the set of directed edges: $E = \{(l_i, l_j) \mid l_i, l_j \in L\}$.

3. T is a set of tuples of the form $\langle d, flag, CD(d), \{EN\}, \{EX\} \rangle$, where d is a tracking device, *flag* indicates whether it is a constrained path (CP) or not, CD(d) is the CD of d, {EN} is a collection of ENLDs, and {EX} is a collection of EXLDs.

4. lb_E is a function lb_E: $E \rightarrow T$ that labels an edge by a tuple from T. An edge $(l_i, l_j) \in E$ is labeled by a tuple $T_{i,j}$ $\langle d_k, CP, CD(d_k)$, EN(l_i, l_j), EX(l_j,

*) $\rangle \in T$ where d_k is a tracking device deployed at location l_j, $CD(d_k)$ is the CD for d_k, EN(l_i, l_j) is an ENLD, and the EXLD for all the out-going locations from l_j is shown as EX(l_j, *). An edge is labeled by a tuple $T_{v_x,j} \in T$ if a virtual tracking device dev_{v_x} is assumed to be deployed at location l_{v_x}. Fig. 5.4 shows an example of the ERDG for the floor plan of Fig. 5.1. Let us consider *edge* (l_1, l_3) where the tuple $T_{1,3}$ is assigned. Regarding the content of $T_{1,3}$, dev_3 is deployed at l_3, CD(dev_3) is the CD for dev_3, EN(l_1, l_3) is the ENLD from l_1 to l_3, and EX(l_3, *) is the EXLD from l_3 to any next destination.



$T_{0,1}$ = (dev_1, CP, CD(dev_1) = 2m, EN(*, l_1) = 4m, EX($l_{1,*}$) = 4m);
$T_{0,2}$ = (dev_2, CP, CD(dev_2) = 2m, EN(*, l_2) = 4m, EX($l_{2,*}$) = 4m);
$T_{1,3}$ = (dev_3, CP, CD(dev_3) = 3m, EN(l_1, l_3) = 8m, EX($l_{3,*}$) = 3m);
$T_{2,3}$ = (dev_3, CP, CD(dev_3) = 3m, EN(l_2, l_3) = 6m, EX($l_{3,*}$) = 3m);
$T_{3,4}$ = (dev_4, CP, CD(dev_4) = 3m, EN(*, l_4) = 5m, EX(l_4,l_5) = 8m, EX(l_4,l_4)=22m);
$T_{4,4}$ = (dev_4, CP, CD(dev_4) = 3m, EX(l_4,l_5) = 8m, EX(l_4,l_4) = 22m);
$T_{4,v1}$ = (dev_{v1}, SCP);
$T_{4,5}$ = (dev_5, CP, CD(dev_5) = 3m, EN(*, l_5) = 3m, EX($l_{5,*}$) = 22m);
$T_{5,5}$ = (dev_5, CP, CD(dev_5) = 3m, EX($l_{5,*}$) = 22m);
$T_{5,v2}$ = (dev_{v2}, SCP); $T_{v1,9}$ = (dev_9, SCP); $T_{v1,10}$ = (dev_{10}, SCP)
$T_{v2,9}$ = (dev_9, SCP); $T_{v1,10}$ = (dev_{10}, SCP)

Fig. 5.4: Graph-based model for CPS

Three mapping structures are defined: location to device-In $L2DIn$: L→D, location to device-Out $L2DOut$: L→2^D, and Device to Location $D2L$: D→L, where L is the set of all locations and D is the set of all tracking devices. For a location l, $L2DIn(l)$ returns the tracking device deployed at l. From the graph it returns the tracking device which is a label for any edge(s) where l is the destination. $L2DOut(l)$ returns all the tracking device which are the labels for the edge(s) where l is the source. These devices are deployed in the adjacent next locations of l. In the third mapping for a tracking device dev, $D2L(dev)$ returns the location of dev, the destination vertex of the edge that

has *dev* in its label. For example in Fig. 5.4, $L2DOut(l_4) = \{dev_4, dev_5, dev_{v1}\}$, $L2DIn(l_4) = dev_4$, and $D2L(dev_4) = l_4$.

Mapping for CPS Locations. In this phase the tracking records are mapped into mapping records. In order to map the $time_{in}$ and $time_{out}$ of an object o at a tracking device *dev* into the $time_{start}$ and $time_{end}$ of o at location l, respectively, the topological information described in the ERDG in Fig. 5.4 is used. The value of $time_{start}$ and $time_{end}$ depend on the speed of o at l. Eq. (5.3), (5.4) and (5.5) are used for deriving the *speed*, $time_{start}$ and $time_{end}$, respectively. In all these equations, $time_{in}$ and $time_{out}$ are taken from tracking records at *L2DIn(l)*. In Eq. (5.3), *CD* (dev_x) represents the *CD* of $L2DIn(l) = dev_x$. As seen in Eq. (5.3), for computing the speed of an object o, the equation divides $CD(dev_x)$ by the time spent by o at dev_x's activation range during its movement through the location $D2L(dev_x)$. In Eq. (5.4) and (5.5), the *ENLD* and *EXLD* depend on where the object is coming from and going to. The value is taken from the tuple $T_{l_{prev},l}$, where l_{prev} is the immediate previous location visited by o before visiting location l. In Eq. (5.4), the duration taken by o for moving through the corresponding *ENLD* of l is calculated by dividing *ENDL* by o's *speed*. Then the calculated duration is subtracted from the corresponding $time_{in}$ for computing $time_{start}$, i.e., the time when o entered at l. Similarly, in Eq. (5.5), the time taken by o for moving through the corresponding *EXLD* of l is calculated by dividing *EXDL* by o's *speed*. Next, the duration is added with the corresponding $time_{out}$ for computing $time_{end}$, i.e., the time when o exited the location l.

$$Speed := \frac{CD(dev_x)}{(time_{out} - time_{in})} \tag{5.3}$$

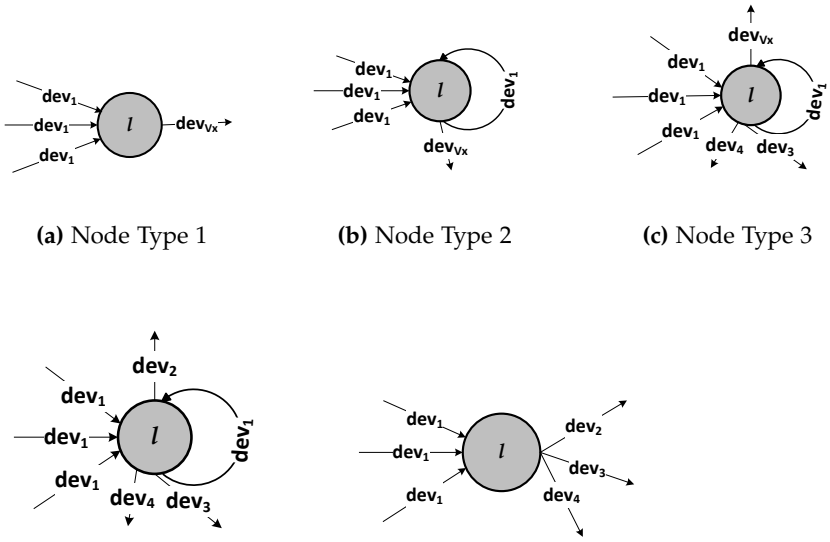
$$time_{start} := time_{in} - \frac{ENLD}{Speed} \tag{5.4}$$

$$time_{end} := time_{out} + \frac{EXLD}{Speed} \tag{5.5}$$

For example, consider the tracking record $\langle o_1, dev_3, 15, 18\rangle$ in Table 5.1. From the graph, *CD* of $dev_3 = CD(dev_3) = 3$ meters and $D2L(dev_3) = l_3$. So the *speed* of o_1 at l_3 is $\frac{3}{18-15} = 1$ meter/time unit. Similarly we can find the $time_{start}$ of o_1 in $D2L(dev_3) = l_3$. The previous record rec_1 says that o_1 was tracked by dev_1 before dev_3. So from the ERDG we need the information from the edge, $E(D2L(dev_1) = l_1, D2L(dev_3) = l_3)$. The ENLD from l_1 to l_3 is $EN\ (l_1, l_3) = 8$m. According to Eq. (5.3) and (5.4), the $time_{start} = 15 - \frac{8m}{3m/(18-15)} = 7$.

If a location contains loop (e.g., l_4, l_5), a tracking device deployed at that location can produce many tracking records for the same object. Thus, this can result in many $time_{out}$s for the same object. Moreover, a location can have a *virtual* destination and also many other regular destinations with readers. Based on the topological connectivity of a location, the nodes of the ERDG

are classified into five types, shown in Fig. 5.5. The node types and the way of deriving the exit time of an object from each of the nodes are explained next.



(a) Node Type 1 **(b)** Node Type 2 **(c)** Node Type 3

(d) Node Type 4 **(e)** Node Type 5

Fig. 5.5: Types of Nodes in CPS

Node Type 1. A location l is of *Node Type 1* if $L2DOut(l) = \{dev_{v_x}\}$. Fig. 5.5a shows an example of *Node Type 1*. As the next location of such a node has no tracking device deployed, an object leaves the location through dev_{v_x} without generating any tracking record. In Eq. (5.5) the $time_{out}$ of an object o_i at this type of location l_i is taken from the tracking record of o_i at $L2DIn(l_i)$ and EXLD $EX(l_i, *)$ is taken from the tuple T_{l_{prev},l_i} of $edge(l_{prev}, l_i)$.

Node Type 2. A location l is of *Node type 2* if $L2DOut(l) = \{L2DIn(l), dev_{v_x}\}$. Fig. 5.5b shows an example of *Node Type 2*. In our example ERDG (Fig. 5.4), l_5 is *Node Type 2*. Here, an object can circulate within the location which generates multiple tracking records and at the end the object leaves the location through dev_{v_x}. In Eq. (5.5), the $time_{out}$ is taken and the *speed* is calculated from the last tracking record of the object from the tracking device of that location. Suppose an object o_2 contains a single record from dev_5: (o_2, dev_5, 36, 39). It means that o_2 did not circulate at $D2L(dev_5) = l_5$ and left the location to any one of the chutes. It is not possible to know when the object actually left l_5. However, we can get the maximum possible value of $time_{end}$ with the help of EXLD from the *edge* (l_5, l_5) which is $EX(l_5, *)$ = 22m and CD

for dev_5 = *CD(*dev_5*)* = 3m. So the $time_{end} = \lceil 39 + \frac{22m}{3m/(39-36)} \rceil = 61$.

Node Type 3. A location l is considered to be *Node Type 3* if $|L2DOut(l)|$ >2 and $\{L2DIn(l), dev_{v_x}\} \subset L2DOut(l)$. Here, an object can circulate in the same location and it can leave the location through dev_{v_x} or other tracking devices. Fig. 5.5c shows an example of *Node Type 3*. In our example ERDG, l_4 is this type of node. During mapping a location of *Node Type 3*, the value of $time_{out}$ from the tracking records is picked in the same way as for *Node Type 2*. If the object has any tracking record from $L2DOut(l) \setminus \{L2DIn(l), dev_{v_x}\}$ (where l is *Node Type 3*), we take the corresponding EXLD; otherwise, we take the EXLD for the loop. For example, for l_4, *L2Dout(*l_4*) = {*dev_4, dev_5, dev_v*}* where dev_4 = *L2DIn(*l_4*)*. As the object o_1 in Table 5.1 has no tracking record from dev_5, the object o_1 should circulate at l_4 and leave the location through dev_{v_x} without generating any tracking record. So, the $time_{end}$ of object o_1 from l_4: $time_{end} = \lceil 54 + \frac{22m}{3m/(54-51)} \rceil = 76$.

Node Type 4 and ***Node Type 5*** do not contain any outgoing edge with dev_{v_x} in label. *Node Type 4* contains a loop, whereas *Node Type 5* does not. Fig. 5.5d and Fig. 5.5e show examples of *Node Type 4* and *Node Type 5*, respectively. In our example ERDG, l_1, l_2, *and* l_3 are *Node Type 5*. As *Node Type 4* contains a loop, during mapping, the value of $time_{out}$ from the tracking records is picked in the same way as for *Node Type 2 and 3*. However, the EXLD *EX(l,* l_{next}*)* is taken from the *edge(*l_{prev}*, l)*. For *Node Type 5*, the $time_{out}$ is directly taken from the tracking record as there is no loop in it. The EXLD in *Node Type 5* is picked in the same way as in *Node Type 4*. In our running example, the $time_{end}$ of o_1 from l_3 is calculated as $time_{end} = \lceil 18 + \frac{3m}{3m/(18-15)} \rceil = 21$.

The algorithm for mapping the tracking records is shown in Algorithm 39. The algorithm takes the tracking records *TR⟨ObjectID, TrackingDeviceID,* $time_{in}$*,* $time_{out}$*⟩* and the graph as input and produce the *mapping table MT⟨ MappingID, ObjectID, LocationID,* $time_{start}$*,* $time_{end}$*⟩* as output. All the tracking records of an object is kept in a variable R in $time_{in}$ ascending order (line 3) and the previous location of the object is initialized to l_0 (i.e., the object is outside of the system) (line 4). For each tracking record $\in R$ of object o_i, the current location is determined by *D2L()* (line 6) and based on the node type of the current location, the necessary values like $time_{out}$, *EXLD*, *ENLD*, and other values are derived (lines 11-33). The variables Δt_{in} and Δt_{out} are used for calculating the speed of the object when it is getting in and getting out, respectively (line 35). The value of Δt_{in} is directly calculated from the first tracking of an object at a location (line 8). However, $\Delta t_{out} = \Delta t_{in}$ for locations of *Node Type 1 and 5* as they do not contain any loops (lines 12, 32). For the locations containing loops *(Node Type 2, 3 and 4)*, the value of Δt_{out} is calculated from the last tracking record of an object in that location (lines 16, 20, 28). As these types of locations may produce multiple tracking records, we consider only the first and last tracking records from the tracking sequence

at such locations and skip the intermediate tracking records (lines 15, 19, 27). After deriving all necessary values, the $time_{start}$ and $time_{end}$ is calculated for the object o_i at l_{curr} (lines 36-37). Finally, the mapping record is inserted into *MT* (line 38). After inserting the mapping records for all the objects, the mapping table *MT* is returned from the algorithm (line 39). Table 5.2 shows the results after mapping from Table 5.1.

Algorithm 4: MappingCPS (TrackingRecords TR, ERDG G) **Result:** *Mapping table MT*

```
1  O := Set of distinct ObjectID∈TR.ObjectID;
2  for each ObjectID o_i ∈ O do
3      R := All records for object o_i from TR in time_in-ascending order;
4      l_prev := l_0;
5      for each record r_i ∈ R do
6          l_curr := G.D2L(r_i.TrackingDeviceID);
7          Inedge := G.E(l_prev, l_curr);
8          Δt_in := r_i.time_out - r_i.time_in;
9          time_in := r_i.time_in;
10         ENLD := Inedge.EN(l_prev, l_curr);
11         if l_curr is Node Type 1 then
12             Δt_out := Δt_in; time_out := r_i.time_out;
13             EXLD := Inedge.EX(l_curr, *);
14         else if l_curr is Node Type 2 then
15             In the loop skip the next subsequent tracking records from R where o_i was tracked by
               r_i.TrackingDeviceID until the last record from the same tracking device;
16             Δt_out := r_last.time_out - r_last.time_in;
17             time_out := r_last.time_out; EXLD := Inedge.EX(l_curr, *);
18         else if l_curr is Node Type 3 then
19             In the loop skip the next subsequent tracking records from R where o_i was tracked by
               r_i.TrackingDeviceID until the last record from the same tracking device;
20             Δt_out := r_last.time_out - r_last.time_in;
21             time_out := r_last.time_out;
22             if o_i has records from any device ∈ G.L2DOut(l_curr) then
23                 EXLD := Inedge.EX(l_curr, l_last+1);
24             else
25                 EXLD := Inedge.EX(l_curr, l_curr);
26         else if l_curr is Node Type 4 then
27             In the loop skip the next subsequent tracking records from R where o_i was tracked by
               r_i.TrackingDeviceID until the last record from the same tracking device;
28             Δt_out := r_last.time_out - r_last.time_in;
29             time_out := r_last.time_out;
30             EXLD := Inedge.EX(l_curr, l_last+1);
31         else if l_curr is Node Type 5 then
32             Δt_out := Δt_in; time_out := r_i.time_out;
33             EXLD := Inedge.EX(l_curr, l_next);
34         CD := Inedge.CD(r_i.TrackingDeviceID);
35         InSpeed := CD/(Δt_in); OutSpeed := CD/(Δt_out);
36         time_start := time_in - ENLD × InSpeed;
37         time_end := time_out + EXLD × OutSpeed;
38         Insert⟨o_i, l_curr, time_start, time_end⟩ into MT;
39 return MT;
```

Table 5.2: Mapping table for Table 5.1 considering CPS

MappingID	ObjectID	LocationID	time$_{start}$	time$_{end}$
map1	o_1	l_1	*2*	*7*
map3	o_1	l_3	*7*	*21*
map5	o_1	l_4	*21*	*76*

3.3 Modeling SCPS

In contrast to CPS, there is no *ENLD*, *EXLD* and *CD* required for SCPS as the tracking devices are deployed in the entry and exit points and speed calculation is unnecessary. However, we also need to consider that there can be multiple entry and exit points in an SCPS location. We use a Reader Deployment Graph (RDG) for modeling the SCPS.

The RDG for SCPS is formally defined by a labeled directed graph $G = (L, E, D, lb_E)$, where:

1. L is the set of symbolic locations where each location is represented as a vertex in G.
2. E is the set of directed edges: $E = \{ (l_i, l_j) \mid l_i, l_j \in L\}$.
3. D is the set of all tracking devices.
4. lb_E represents a labeling strategy of an edge by tracking device(s) from D. An edge $(l_i, l_j) \in E$ is labeled by a set of tracking devices $D_k \subseteq D$ if D_k are deployed between location l_i and l_j and an object must be detected by any device from D_k if it goes from l_i to l_j. Fig. 5.6 shows the RDG for the floor plan of Fig. 5.2

Mapping for SCPS Locations. We define a mapping function $Dest: \{l, d\} \rightarrow l'$ where $l, l' \in L$ and $d \in D$. $Dest(l, d)$ returns node l' from G where d is the label for the edge $E\ (l, l') \in E$. It means that tracking device d is deployed between location l and l' and if an object goes from l to l' then it will be detected by d. For this mapping, each tracking record of an object o is accessed and with the help of the *Dest* function it is determined where o is entering. Then the $time_{in}$ in the tracking record becomes $time_{start}$ and the $time_{in}$ of the next tracking record becomes $time_{end}$ for the entered location. For example, let Table 5.1 be constructed from the movement in the floor plan of Fig. 5.2. From rec_1 we see that o_1 was tracked by dev_1 from time 4 to 5. The initial location of o_1 is considered as l_0. From the graph in Fig. 5.6, $Dest(l_0, dev_1) = l_1$. From the tracking record, $time_{start} = time_{in} = 4$. However, the $time_{end} = time_{in}$ from rec_3 is 15. So, the mapping record says that o_1 was at l_1 from time 4 to 15.

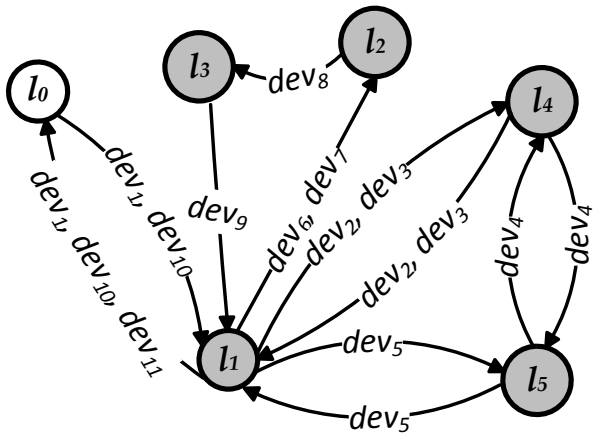


Fig. 5.6: Graph-based model for SCPS

Algorithm 10 shows the steps of mapping the tracking records collected from an SCPS. It takes tracking records *TR* and RDG *G* as input and produces the mapping table *MT* for the given tracking records. For each object o_i, it takes the tracking records into *R* sorted in $time_{in}$-ascending order (line 3). In the beginning, the current location of the object is initialized to l_0 (line 4). Then for each tracking record of o_i, the algorithm finds the entered location using the *Dest* function with the help of *G*, and takes the $time_{in}$ as the $time_{start}$ (lines 5-7). The $time_{end}$ is taken from the $time_{in}$ of next record as it ensures that the object left the location after generating the next record (line 8). Finally, a mapping record is inserted into *MT* from the generated information (line 9). Table 5.3 shows the SCPS mapping results for Table 5.1.

4 Efficient Dense Location Extraction

The dense locations can now be extracted from the mapping records in the *mapping table*. As seen, a mapping record contains the entire stay of an object inside a location. For a CPS location like baggage on a conveyor belt, it is not common that an object re-visit a location after visiting another location, whereas it is common for an SCPS location. Moreover, the stay duration of an object in a CPS location is always more or less the same, e.g., all bags spend the same time at the sorter. In our setting, an object visiting a location multiple times is treated as multiple objects for density computation for that

Algorithm 5: MappingSCPS(TrackingRecords TR, RDG G) **Result:** *Mapping table MT*

```
1  O := Set of distinct ObjectID from TR.ObjectID;
2  for each ObjectID o_i ∈ O do
3      R := All records for o_i from TR in time_in-ascending order;
4      Location l:= l_0;
5      for each record r_i ∈ R do
6          l := G.Dest(l, r_i.TrackingDeviceID);
7          time_start := r_i.time_in;
8          time_out := r_{i+1}.time_in (i < |R|);
9          Insert⟨o_i, L, time_start, time_end⟩ into MT;
10 return MT;
```

Table 5.3: Mapping table for Table 5.1 considering SCPS

MappingID	ObjectID	LocationID	$time_{start}$	$time_{end}$
map1	o_1	l_1	*4*	*15*
map3	o_1	l_4	*15*	*26*
map5	o_1	l_5	*26*	*51*
map8	o_1	l_4	*51*	...

location since reappearance contributes to the density. However, the duration based density (DBD) makes it more general for broader application scenarios as the length of the stay of different objects also contribute to the density in this case.

4.1 Dense Location Queries (DLQ)

A dense location query $DLQ[q_s, q_e, \theta]$ finds the dense locations between time q_s and q_e where θ is the density threshold. In the inner part of a *DLQ*, there is a density query, a count query (CQ) for *CBDLQ*, a duration query (DQ) for *DBDLQ*, and a mapping record query (RQ). Fig. 5.7 and 5.8 show the naive approaches for processing a *CBDLQ* and a *DBDLQ*, respectively. When a $CBDLQ[q_s, q_e, \theta]$ is triggered, the system issues a count based density query $CBDQ[q_s, q_e]$, the $CBDQ[q_s, q_e]$ then issues a $CQ[q_s, q_e]$ which issues an $RQ[q_s, q_e]$. The RQ gets the mapping table from the database and returns the mapping records that falls between the times q_s and q_e. From the relevant mapping records, the CQ counts the number of objects at each location. Next, the *CBDQ* computes the density of each location with the result from CQ and the capacity of the corresponding location. Finally, the *CBDLQ*

returns the locations with *density* $> \theta$. However, for a $DBDLQ[q_s, q_e, \theta]$ described in Fig. 5.8, there is no CQ. In this case a duration query $DQ[q_s, q_e]$ is processed which takes the mapping records for the interval $[q_s, q_e]$ and computes the total length of stay of the objects at each location within the query time interval. Then the $DBDQ[q_s, q_e]$ uses this duration information and the capacities of the locations to compute the density of each location.

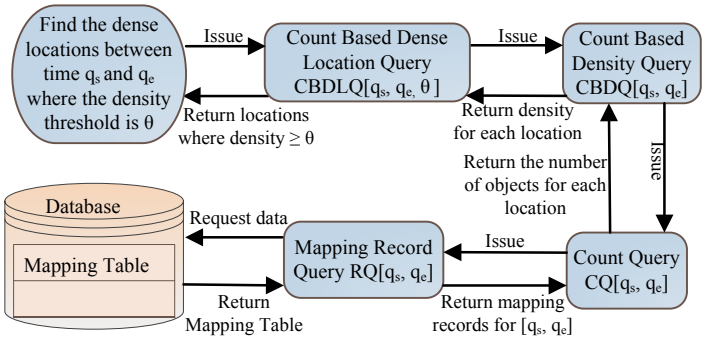


Fig. 5.7: Query steps for count based dense location query (CBDLQ)

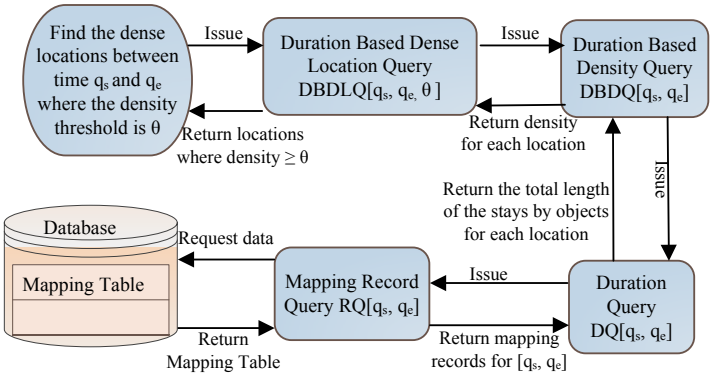


Fig. 5.8: Query steps for duration based dense location query (DBDLQ)

A DLQ has to access aggregate information for a given time interval from a large amount of data from the mapping table. We now present an indexing technique to facilitate that. Its temporal part is inspired by the *time index* described in [31]. Instead of indexing all the records of the mapping table, we index all the intervals of each location using separate trees and store aggregate values instead of pointers to the leaf nodes. Moreover, unlike the B^+-Tree, we link the nodes at intermediate levels. All these improvements enable us to avoid accessing detailed data records and further offer significant

pruning opportunities. This new index structure is called the *Dense Location Time Index* (*DLT*-Index).

Table 5.4: Example mapping records at location l_1

MappingID	ObjectID	LocationID	time$_{start}$	time$_{end}$
r_1	o_1	l_1	2	7
r_2	o_2	l_1	5	11
r_3	o_{18}	l_1	8	20
r_4	o_{15}	l_1	9	18
r_5	o_{16}	l_1	11	20
r_6	o_{12}	l_1	67	70
r_7	o_{19}	l_1	22	26
r_8	o_{20}	l_1	24	29

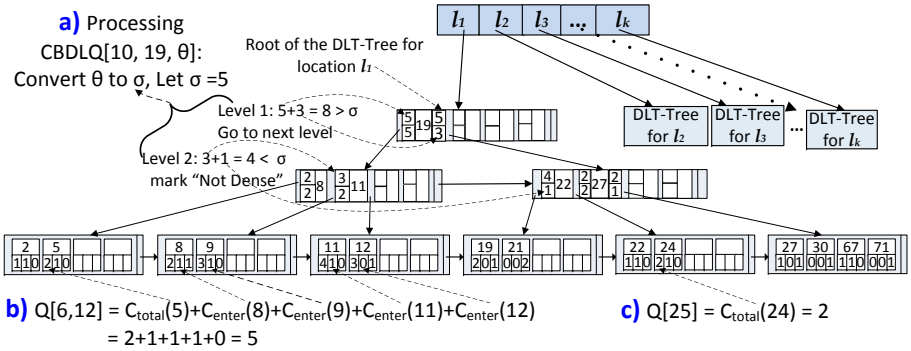


Fig. 5.9: The *DLT*-Index with the *DLT*-Tree for location l_1 of Table 5.4. The figure also shows three examples of processing a) CBDLQ, b) interval query, and c) point query over the *DLT*-Index.

4.2 The DLT-Index

In the *DLT*-Index we maintain a separate tree, called a *DLT*-Tree, for each location. As a result, the *DLT*-Index is a collection of *DLT*-Trees. Let us consider a set of mapping records at location l_1 shown in Table 5.4. Fig. 5.9 shows an example of *DLT*-Index with the *DLT*-Tree constructed for the data given in Table 5.4. An entry of a *DLT*-Tree contains a time point with some aggregate information of objects at that time point. In the *DLT*-Tree of a location l, each leaf node entry at time point t_i is of the form: $\langle t_i, C_i \rangle$, where $C_i \langle c_{total}, c_{enter}, c_{exit} \rangle$ is a tuple with some aggregate information of the objects at l valid during $[t_i, t_i^+)$, where t_i^+ is the next indexed time point after t_i, c_{total} is the total number of objects present at l during t_i, c_{enter} is the total number

of objects that entered l during t_i, and c_{exit} is the total number of objects that exited l during t_i-1. Besides, each non-leaf entry at time point t_i contains a tuple $C_{nl}(t_i)$ which is of the form:$\langle c'_{total}, c'_{enter} \rangle$ and their values are described as follows:

1. For the left-most entry at a level: c'_{total} = c'_{enter} = the total number of objects that entered l until t_i-1.

2. For the entries other than the left-most entry: c'_{total} is the total number of objects present at l during interval $[t_i^-, t_i)$ and c'_{enter} is the total number of objects that entered l during interval $[t_i^-, t_i)$, where t_i^- is the immediate left entry of t_i at the same level.

In addition to C_{nl}, the right-most entry t_i of each non-leaf level also contains another tuple $C_r \langle c''_{total}, c''_{enter} \rangle$, where c''_{total} is the total number of objects present at l during t_i to the max time stamp in the tree and c''_{enter} is the total number of objects that entered l during t_i to the max timestamp in the tree.

The tree construction and insertion of a new entry in the *DLT*-Tree is very similar to their counterparts for the B^+-Tree. The time points are the keys and the aggregate information C are the data values. The value of C at the leaf levels can be precomputed or can be computed while inserting. After constructing the tree, the aggregate information of the non-leaf entries and the links between the non-leaf nodes have to be established.

4.3 Tree Construction from Historical Data

During the tree construction, the historical data are indexed. Let the set of all intervals available in the data set be, $I = \{I_1, I_2, ..., I_n\}$. For an interval I_i, the value of $I_i.t_s$ and $I_i.t_e$ represents the start time and end time, respectively. Additionally, the value of $I_i^{\dashv} = I_i.t_e + 1$ represents the timestamp after $I_i.t_e$. The *DLT*-Index has to index all the time points of P that is defined as follows:

$$P = \{t_i | \exists I_j \in I((t_i = I_j.t_s) \vee (t_i = I_j^{\dashv}))\}$$

For example, considering Table 5.4, the points that need to be indexed are *P = {2, 5, 8, 9, 11, 12, 19, 21, 22, 24, 27, 30, 67, 71}*. As seen for *MappingID* $r1$, the $time_{end}$ = 7 is not included in P as we index the timestamp after 7 which is 8. Also at time point 8, o_{18} has entered the location (*MappingID* $r3$). Before the tree construction, all the time points of P are sorted in ascending order into P_s. Each of the time points $t_i \in P_s$ additionally contains a C_i where c_{enter} and c_{exit} are directly known while getting each element of P_s from the data set. For example, at the time point 8: c_{enter} = 1 and c_{exit} = 1; at time point 21: c_{enter} = 0 and c_{exit} = 2 as two objects have $time_{end}$ at 21-1 = 20. The c_{total} at time point t_i is calculated by Eq. (5.6). For example, at the initial stage C_0 =

$\langle 0,0,0 \rangle$. For the first time point 2, c_{enter}=1 and c_{exit} = 0. So, c_{total} = 0+1-0=1. As a result, C_1 = $\langle$1,1,0$\rangle$. Similarly for time point 5, c_{total} = 1+1-0 = 2 and C_2 = $\langle$2,1,0$\rangle$. The complete list of C_is is shown in the leaf nodes of the *DLT*-Tree for the location l_1 in Figure 5.9.

$$c_{total}(t_i) := c_{total}(t_{i-1}) + c_{enter}(t_i) - c_{exit}(t_i) \tag{5.6}$$

The insertion of the time points is done in the same way as for a B^+-Tree, where the time points are the keys and C are the values. However, for the entries of the intermediate levels the value of C_{nl} has to be calculated. Maintaining the aggregate information in the leaf nodes gives advantage for such calculations. For any non-leaf entry t_i, if it is the left-most entry at its level, the C_{nl} is calculated by Eq. (5.7). In the equation, the value of c_{total} from the left-most entry t_1 at the leaf level is added with all the c_{enter} after t_1 until the immediate left entry of t_i at the leaf level.

$$C_{nl}(t_i).c'_{total} := C_{nl}(t_i).c'_{enter} := c_{total}(t_1) + \sum_{j=2}^{i-1} c_{enter}(t_j) \tag{5.7}$$

Besides, for any non-leaf entry t_i, if it is not the left-most entry in its level, the C_{nl} is calculated by Eq. (5.8) and (5.9). Let t_i^- be the entry immediately before t_i at its level and k be the position of t_i^- in the leaf node. Then,

$$C_{nl}(t_i).c'_{total} := c_{total}(t_k) + \sum_{j=k+1}^{i-1} c_{enter}(t_j) \tag{5.8}$$

$$C_{nl}(t_i).c'_{enter} := \sum_{j=k}^{i-1} c_{enter}(t_j) \tag{5.9}$$

Similarly, for the right-most entry t_i at a non-leaf level, C_r is calculated by Eq. (5.10) and (5.11) as follows:

$$C_r(t_i).c''_{total} := c_{total}(t_i) + \sum_{j=i+1}^{max} c_{enter}(t_j) \tag{5.10}$$

$$C_r(t_i).c''_{enter} := \sum_{j=i}^{max} c_{enter}(t_j) \tag{5.11}$$

In Eq. (5.10) and (5.11), t_{max} represents the maximum value of the indexed time point at the leaf level.

For example in Fig. 5.9 at level 2, 8 is the left-most entry. So $C_{nl}(8).c'_{total}$ = $C_{nl}(8).c'_{enter}$ = $c_{total}(2)$ + $c_{enter}(5)$ = 1+1 = 2. Considering 22 at level 2 which

is not the left-most entry, $C_{nl}(22).c'_{total}$ = $c_{total}(11)$ + $c_{enter}(12)$ + $c_{enter}(19)$ + $c_{enter}(21)$ = 4+0+0+0 = 4. In the example tree, 27 is the right-most entry of level 2. The value of $C_r(27).c''_{total}$ = $c_{total}(27)$ + $c_{enter}(30)$+ $c_{enter}(67)$ +$c_{enter}(71)$ = 1+0+1+0 = 2.

The algorithm for *DLT*-Index construction from historical data is shown in Algorithm 12. The algorithm takes mapping records R and a set of locations L as input and creates the *DLT*-Index as the result. For each location, all the $time_{start}$ and $time_{end}$+1 is taken from the data set and their union is stored into P (lines 1-4). The value of $C_{total}(P_0 \notin P)$ is initialized to zero for handling the special case for the first time point P_1 (line 4). Then for each time point $P_i \in P$, the values of C_{enter}, C_{exit} and C_{total} are computed and the tuple C for P_i is formed (lines 5-9). The tuple C is then inserted into the *DLT*-Tree for the corresponding location like B^+-Tree insertion, where P_i is the key and C is the data value (line 10). After inserting all the values of the particular location, the intermediate links within the nodes in the non-leaf levels are established and the values of C_{nl} and C_r are filled in the *DLT*-Tree (lines 11-12).

Algorithm 6: TreeConstruction (MappingRecords R, Locations L) **Result:** *DLT-Index*

1 **for** *each location* $l_i \in L$ **do**
2 $\quad I_s$:= All $time_{start}$ of l_i from R;
3 $\quad I_e$:= All $time_{end} + 1$ of l_i from R;
4 $\quad$ time points $P := I_s \cup I_e$; $C_{total}(P_0 \notin P) := 0$;
5 $\quad$ **for** *each time point* $P_i \in P$ **do**
6 $\quad\quad C_{enter}(P_i)$:= Number of objects containing P_i as $time_{start} \in I_s$;
7 $\quad\quad C_{exit}(P_i)$:= Number of objects containing P_i-1 as $time_{end} \in I_e$;
8 $\quad\quad C_{total}(P_i) := C_{total}(P_{i-1}) + C_{enter}(P_i) - C_{exit}(P_i)$;
9 $\quad\quad C := \langle C_{total}(P_i), C_{enter}(P_i), C_{exit}(P_i) \rangle$;
10 $\quad\quad$ Insert into *DLTIndex*[l_i] just like B^+-Tree insertion method with P_i as key and C as a data value;
11 $\quad$ Create intermediate links within the nodes at the non-leaf levels ;
12 $\quad$ Fill the values of C_{nl} and C_r for each entry at the non-leaf levels;

4.4 Updating *DLT*-Index

In an update, a new time interval is to be inserted and the interval is not necessarily greater than the existing time points of the tree. Here, a new entry can be added between existing time points that can result in inconsistencies in the aggregate information in some other entries. As a result, it requires an adjustment operation for recalculating the aggregate information for the affected entries. For any interval $I[t_s, t_e]$, the update algorithm has to insert

$I.t_s$ and $I^{\dashv} = I.t_e + 1$ separately. Initially, the time points $I.t_s$ and $I^{\dashv}$ contains tuple $\langle 1, 1, 0 \rangle$ and $\langle 0, 0, 1 \rangle$, respectively. If these two points do not already exist, they are added in the tree just like B^+-Tree insertion. However, if $I.t_s$ already exists in the tree, it increases the corresponding c_{total} and c_{enter} by 1; and if $I^{\dashv}$ already exists, it decreases c_{total} and increases c_{exit} by 1. Finally, the c_{total} for each of the points in the leaf node entries that fall in the interval $I[t_s, t_e + 1]$ need to be adjusted according to Eq. (5.6). Similarly, the values of C_{nl} for each of the non-leaf entry that fall in the interval $I[t_s, t_e^{>}]$ need to be adjusted, where $t_e^{>}$ is the immediate next entry after t_e at each level if t_e is not present at that level, otherwise $t_e^{>} = t_e$.

5 Query Processing

Now we discuss how the aggregate point, interval, and duration queries are processed by using our *DLT*-Index. Subsequently, we will show the *CBDLQ* and *DBDLQ* processing with and without pruning.

5.1 Aggregate queries

There are three types of aggregate queries: a) point queries, b) interval queries, and c) duration queries. A point query finds the total number of objects at a particular time *point*. An interval query finds the total number of objects that appeared in a particular time *interval*. A duration query finds the total stay duration of objects in a particular time interval. For processing a point query $Q[q_t]$, a B^+-Tree like search is performed for finding the appropriate leaf node for q_t. Then it finds the last entry $t_a \leq q_t$ in the node and returns $C_{total}(t_a)$ as the result. For example, consider a point query $Q[25]$. The B^+-Tree like search will find the node shown in Fig 5.9c. From the resulting node, entry 24 is the last entry which is ≤ 25. So the query will return $c_{total}(24) = 2$. Conversely, for an interval query $Q[q_s, q_e]$, it will first process a point query $Q[q_s]$ and take $C_{total}(t_a)$. However, instead of returning the result, it continues the processing and finds the last entry $t_b \leq q_e$ from the leaf nodes. For this purpose, it may have to access consecutive leaf nodes one after another. If $t_a = t_b$, it returns $C_{total}(t_a)$ as the result. Otherwise, it adds all the C_{enter} for each entry after t_a until t_b. For example, consider an interval query $Q[6, 12]$. The B^+-Tree search will find the leaf node containing $\{2, 5\}$ as shown in Fig. 5.9b. Then it finds the entry 12 which is the last entry ≤ 12. So the calculation of the result will be: $Q[6, 12] = c_{total}(5) + c_{enter}(8) + c_{enter}(9) + c_{enter}(11) + c_{enter}(12)$ = 2+1+1+1+0 = 5. The processing of an interval query is shown in Algorithm 5. The processing steps of a duration query will be discussed in Section 5.3.

Algorithm 7: IntervalQuery (Location l, Start time q_s, End time q_e) **Result:** ***Total number of objects during*** $[q_s, q_e]$ ***at*** l

```
1 root : = DLT-Tree for location l;
2 Leaf node node : = B+-Tree search (q_s) and find the leaf node
  containing q_s;
3 Find the last entry t_a from the node where t_a ≤ q_s;
4 n : = C_total(t_a)+Σ_{i=b}^{c} c_enter(t_i), where t_b is the immediate indexed time
  point after t_a where t_a ≤ q_e and t_c is the last indexed time point ≤ q_e ;
5 return n;
```

5.2 Count Based Dense Location Query (CBDLQ)

The naive approach of processing a $CBDLQ[q_s, q_e, \theta]$ over the DLT-Index is to just get the results of an interval query $Q[q_s, q_e]$ for each of the locations, compute the density from the results, and then determine which locations are dense. However, in the naive approach, the query has to access the leaf nodes and then compute the aggregate results by accessing all the leaf level nodes that cover the query interval. The access of the nodes at the leaf level as well as other levels can be reduced by our pruning technique. As described in Section 4.2, the DLT-Index contains some aggregate information in the non-leaf node entries for pruning purpose. We first introduce the following two observations.

Observation 1: If the number of objects in an interval i_1 is less than a threshold k, a smaller interval i_2 that is fully covered by i_1 must have less than k objects.

Observation 2: If the number of objects in an interval i_1 is greater than a threshold k, a larger interval i_2 that fully covers i_1 must have more than k objects.

In our pruning technique, the first observation helps prune at the non-leaf levels, whereas the second observation helps prune at the leaf level. Note that the cases covered by observation 1 and observation 2 are mutually exclusive. In a $CBDLQ[q_s, q_e, \theta]$, the given θ is a density threshold rather than a number of objects. Before starting the query processing, we convert θ to a number of objects threshold σ_i for each location l_i. The value of σ for a location l_i for a $CBDLQ[q_s, q_e, \theta]$ is derived from the density formula Eq. (5.1) as shown in Eq. (5.12). In Eq. (5.12), $\Delta t = q_e - q_s$.

$$\sigma(l_i, \theta) := \Delta t \times capacity(l_i) \times \theta \tag{5.12}$$

While processing a $CBDLQ[q_s, q_e, \theta]$, the value of σ for each location has to be calculated. The query has to traverse the DLT-Tree of each location. The way of accessing the next level is similar to a B^+-Tree search for q_s. At

each non-leaf level $level_i$ of the DLT-Tree of a location l_i, the smallest interval $[t_a, t_b)$ that can cover $[q_s, q_e]$ has to be found. After that, the total number of objects $n(level_i)$ during $[t_a, t_b)$ is calculated from the aggregate information stored in all the entries between t_a and t_b at level $level_i$. The value of $n(level_i)$ is compared with σ. If $n(level_i) < \sigma$, l_i is marked as 'Not Dense' based on *observation 1* and further processing of the tree is avoided. However, if $n(level_i) \not< \sigma$, the next level is accessed and further processing is done in a similar way to the other levels. If the query reaches the leaf level, it starts calculating the total number of objects during $[q_s, q_e]$. During the calculation, at each step it compares the sum with σ and if $sum \geq \sigma$ then based on *observation 2* the location l_i is marked as *'Dense'*. As a result, it skips the access of the next entries and nodes and avoids further calculation. However, if the query processing is not skipped, the full sum is calculated to decide whether l_i is dense or not.

Consider Fig. 5.9a, where the processing of *CBDLQ[10, 19, θ]* is shown for location l_1. Let $\sigma = 5$ be derived from θ. At the root level, the smallest interval that covers $[10, 19]$ is [*min, max*), where *min* and *max* are the starting and ending time points of the tree, respectively. For *min*, the $C_{nl}.c'_{total}$ from the left-most entry is taken, and for *max*, $C_r.c''_{enter}$ is taken from the right-most entry of the current level. So in our case the total number of objects during this period is $C_{nl}(19).c'_{total}$ + $C_r(19).c''_{enter}$ = 5+3=8. As 8 $\not< \sigma$, the query has to go to the next level based on a B^+-Tree like search for 10. At this level $[8, 22)$ is the smallest interval that covers $[10, 19]$. Now the total number of objects for the period is $C_{nl}(11).c'_{total}$ + $C_{nl}(22).c'_{enter}$ = 3+1=4. As $4 < \sigma$, the location l_1 is not dense during $[10, 19]$. As a result, it does not need to access the next level and do further processing for l_1. Now consider another query $CBDLQ[6, 12, \theta]$ and let $\sigma = 3$. During the processing of this query on the DLT-Tree for l_1 shown in Fig. 5.9, the pruning with *observation 1* does not work and the query will access the leaf level. However, the query does not have to fully compute the total number of objects during $[6, 12]$ as the sum up to time point 9 is: 2+1+1=4 (Fig. 5.9b) and $4 \geq \sigma$. So, it can be deduced that the location is dense during time interval $[6, 12]$ without further processing.

The processing of a CBDLQ including our pruning strategy is shown in Algorithm 27. It takes the set of locations, a time interval and a threshold θ as input, and returns the CBDLs for the given time interval. For each location l_i, $\sigma(l_i, \theta)$ is calculated and the root of DLT-Tree is assigned into T (lines 1-2). For each non-leaf level of T, it finds the first entry $t_a > q_s$ and $t_b > q_e$. Then, the total number of objects n during $[t_a^-, t_b)$ is calculated from the aggregate information at the current level (lines 4-11), where t_a^- is the entry before t_a. If t_a is the left-most entry, then $t_a^- := min$, where *min* is the smallest time point in the tree. If both q_s and q_e are larger than or equal to the right-most entry t_b, the total number of objects during $[t_b, max]$ is taken from the $C_r(t_b).c''_{total}$ (lines 5-6). After computing n, it is compared with σ and if the location is

Algorithm 8: CBDLQ (StartTime q_s, EndTime q_e, Threshold θ, Locations L) **Result:** ***CBDLs***

```
1  for each location l_i ∈ L do
2      σ := σ(l_i, θ); T := DLT-Tree for l_i;
3      for each level level_i of T (B+-Tree like level access for q_s) do
4          if level_i is non-leaf level then
5              if q_s, q_e ≥ the right-most entry t_b of level_i then
6                  n = C_r(t_b).c''_total; Go to line 12;
7              t_a := first entry of level_i which is > q_s;
8              t_b := first entry of level_i which is > q_e;
9              n := C_nl(t_a).c'_total + Σ_{j=a+1}^{b} C_nl(t_j).c'_enter;
10             if t_b is the right-most entry of level_i then
11                 n += C_r(t_b).c''_enter;
12             if n < σ then
13                 l_i.status := "Not Dense"; //Observation 1
14                 break;
15         else
16             t_a := last entry in the current node which is ≤ q_s;
17             n := c_total(t_a);
18             for each entry t_b > t_a ∧ t_b ≤ q_e in the leaf level do
19                 n += c_enter(t_b);
20                 if n ≥ σ then
21                     l_i.status := "Dense"; //Observation 2
22                     break;
23             if n ≥ σ then
24                 l_i.status := "Dense";
25             else
26                 l_i.status := "Not Dense";
27 return the locations ⊆ L that are marked as "Dense";
```

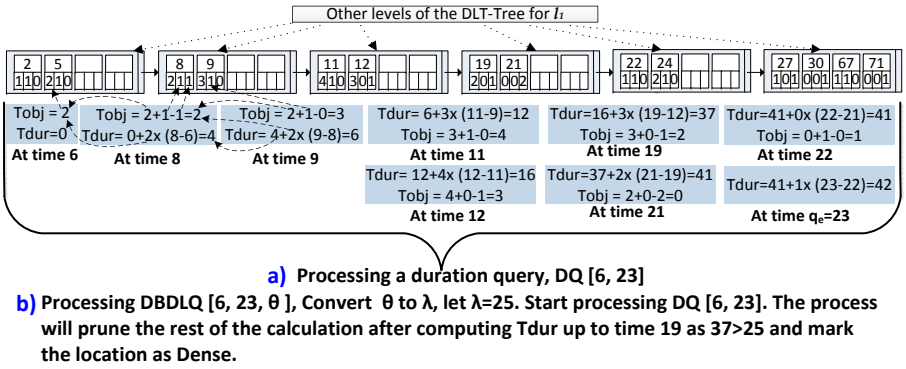
not going to be a dense location, it is marked as 'Not Dense' and the query stops further processing for the current location (lines 12-14). If the query processing is not pruned at any non-leaf level, at the leaf level it computes the total number of objects n during $[q_s, q_e]$ (lines 15-22). First, it finds the last entry t_a in the current node where $t_a \leq q_s$ and takes $c_{total}(t_a)$ (lines 16-17). Then, in a loop it starts adding c_{enter} from each consecutive entry until t_b where $t_b > t_a$ and $t_b \leq q_e$. However, after adding information from each entry

into n, it checks whether $n \geq \sigma$ or not. If the condition is true (i.e., $n \geq \sigma$), the location is marked as 'Dense' and further calculation of n is pruned (lines 20-22). Otherwise, the final value of n is checked to determine whether the location is dense or not (lines 23-26). For the tree of each of the locations, the algorithm iterates according to the process described above. Finally, the algorithm returns all the locations that are marked as 'Dense' (line 27).

5.3 Duration based dense location query (DBDLQ)

In a $DBDLQ[q_s, q_e, \theta]$, the total length of stay by the objects within the interval $[q_s, q_e]$ is computed from the stored aggregate information in the leaf nodes of the DLT-Tree of each location. Before discussing the processing of a DBDLQ, we first discuss the processing of a duration query $DQ[q_s, q_e, l_i]$ using our DLT-Index. The processing steps of a DQ are shown in Algorithm 15. The algorithm takes the query start time q_s, end time q_e and location l_i as input and returns the total length of the stay T_{dur} by the objects at l_i during the given time interval. During processing of a DQ, we incrementally maintain the sum of the stay duration of objects between the query start time q_s until the accessed time point at the leaf level (lines 7-12). At first, a set of variables including T_{dur} are initialized to zero (line 2). Variable T_{obj} is used to track the number of objects staying between the time points, and variable *EndFlag* is used to keep track of whether the tree has q_e as an entry in a leaf node. A B^+-Tree like search for q_s is performed to access the appropriate leaf node in the DLT-Tree for the location l_i (line 3). From the leaf node, the last entry $\leq q_s$ is assigned to t_a (line 4). A variable t_{prev} is initialized with q_s to store the previously accessed time point for computing the duration when accessing a time point at the leaf level (line 5). For each entry t_b (where $t_b > t_a \wedge t_b \leq q_e$) at the leaf level, the aggregate information is accessed and T_{dur} is computed up to the accessed time point (line 8). Then the value of T_{obj} is updated with the new information found at that time point and will be used for the next iteration (line 9). During the final iteration of the loop, if $t_b = q_e$, *EndFlag* is set to 1 to indicate that the T_{dur} is fully computed (lines 11-12). Otherwise, T_{dur} is updated with the remaining time value (i.e., t_{prev} to q_e) that was not added during the loop (lines 13-14). Finally, the final value of T_{dur} is returned from the algorithm (line 15). From our DLT-Tree, we only know how many objects entered, exited, and stayed in the location at different time points. As a result, during the computation of the total duration, we cannot figure out exactly which object entered or exited. However, as we incrementally maintain the sum of the stays by the objects from the aggregate information and carefully consider an increase and decrease in the number of objects at each time point during the query time interval, the total duration computed by our algorithm will remain the same.

An example of processing a duration query *DQ[6, 23,* l_1*]* is shown in



a) Processing a duration query, DQ [6, 23]

b) Processing DBDLQ [6, 23, θ], Convert θ to λ, let λ=25. Start processing DQ [6, 23]. The process will prune the rest of the calculation after computing Tdur up to time 19 as 37>25 and mark the location as Dense.

Fig. 5.10: Processing DQ and DBDLQ using the *DLT*-Index

Fig. 5.10a. The figure only shows the leaf level of the tree. After performing a $B+$-Tree like search for 6, the query finds the leaf level node containing {2, 5}. As the location already contains two objects at time 6, the value of T_{obj} is initialized with 2. The process subsequently accesses the next entries and updates the value of T_{obj} and T_{dur} with the help of the aggregate information, and the other values that are computed until the previous time point. As there is no entry for 23 in the leaf nodes, the remaining duration is added from the remaining objects and the computed value until time 22.

The naive approach processes a *DBDLQ*$[q_s, q_e, \theta]$ by processing a *DQ*$[q_s, q_e]$ for each of the locations, computing the density using Eq. (5.2) with the help of the stay duration from *DQ*, and then determines which locations are dense based on the threshold θ. Similar to *Observation 2* that is used for pruning at the leaf level for a *CBDLQ*, we have another observation for *DBDLQ* for pruning.

Observation 3: If the total length of the stay duration of the objects in an interval i_1 is greater than a threshold k, the total length of the stay duration of the objects in a larger interval i_2 that fully covers i_1 must also be greater than k.

While processing a *DBDLQ*, *Observation 3* helps prune a number of node accesses at the leaf level if the corresponding location is dense. Like σ used in the *CBDLQ*, we convert the density threshold θ into a duration threshold λ. The duration threshold λ is derived from Eq. (5.2) as shown in Eq. (5.13). In Eq. (5.13), $\Delta t = q_e - q_s$.

$$\lambda(l_i, \theta) = \Delta t \times capacity(l_i) \times \theta \tag{5.13}$$

For a $DBDLQ[q_s, q_e, \theta]$, if the total length of the stays of the objects during the interval $[q_s, q_e]$ at location l_i is greater than or equal to $\lambda(l_i, \theta)$, the location l_i is marked as 'Dense'. While processing a $DBDLQ[q_s, q_e, \theta]$, a du-

Algorithm 9: DQ (StartTime q_s, EndTime q_e, Location l_i) **Result:** ***Total stay duration by objects at*** l_i ***during*** $[q_s, q_e]$

```
1  T := DLT-Tree for l_i;
2  T_dur := 0; T_obj := 0; EndFlag := 0;
3  node := leaf level node of T containing q_s; //Obtained by B+-Tree like
   search for q_s
4  t_a := last entry in node which is ≤ q_s;
5  t_prev := q_s;
6  T_obj := c_total(t_a);
7  for each entry t_b>t_a ∧ t_b≤q_e in the leaf level do
8      T_dur := T_dur + T_obj × (t_b - t_prev) ;
9      T_obj :=T_obj + c_enter(t_b) - c_exit(t_b);
10     t_prev := t_b ;
11     if t_b = q_e then
12         EndFlag := 1;
13 if EndFlag = 0 then
14     T_dur := T_dur + T_obj × (q_e - t_prev) ;
15 return T_dur;
```

ration query $DQ[q_s, q_e, l_i]$ is performed for each location with an additional condition on the value of T_{dur}, described as follows. During the processing of the DQ, at each step the incremented value of T_{dur} is compared with λ. If the query finds the current value of $T_{dur} \geq \lambda$, the corresponding location is marked as 'Dense', and further processing for the location is pruned. For example, in Fig. 5.10b the processing of $DBDLQ[6, 23, \theta]$ is shown. Suppose after converting θ to λ the value of λ is 25. In this case, the query has to compute until time point 19 as the value of T_{dur} =37 $\geq$ 25. As a result, the query does not need to calculate the value of T_{dur} further, and the location is marked as 'Dense' according to *Observation 3*.

The processing of a $DBDLQ$ including our pruning strategy is shown in Algorithm 9. The algorithm takes a query interval $[q_s, q_e]$, a density threshold θ and a set of locations L as input, and returns the set of dense locations $\subseteq L$ as the result. For each location l_i, the value of λ is calculated (line 2) and a duration query $DQ[q_s, q_e, l_i]$ is processed with some additional condition in its steps (line 3). During processing the DQ, the new value of T_{dur} at each step is compared with λ. If $T_{dur} \geq \lambda$, the location is marked as 'Dense' and further processing about the location is skipped (based on *Observation 3*). However, if the processing is not skipped, based on the final value of T_{dur}, it is determined whether the location is dense or not (lines 4-8). Finally, all the dense locations are returned as the result of the query (line 9).

Algorithm 10: DBDLQ (StartTime q_s, EndTime q_e, Threshold θ, Locations L) **Result:** ***Set of dense locations***

```
1 for each location l_i ∈ L do
2     λ := λ(l_i, θ) ;
3     Conditionally process DQ[q_s, q_e, l_i] and after computing T_dur at each
      step (see line 8 of Algorithm 15) compare it with λ. If at any step
      T_dur ≥ λ then mark l_i.status := "Dense" and skip further processing
      for l_i;
4     If the processing for l_i was not skipped in the above steps then
      compare the final value of T_dur with λ.
5     if T_dur ≥ λ then
6         l_i.status := "Dense";
7     else
8         l_i.status := "Not Dense";
9 return the locations ⊆ L that are marked as "Dense";
```

6 Experimental Study

6.1 Experimental Setup

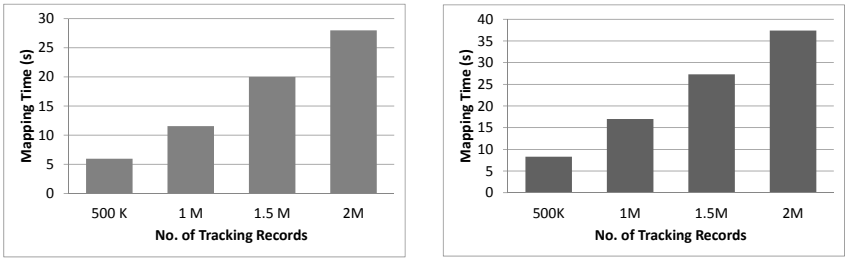
The mappings for both CPS and SCPS are implemented and the aggregate information from the mapping records are precomputed using C#. The *DLT*-Index is implemented using C. A leading RDBMS is used for all SQL queries, that are compared with. The experiments are conducted on a laptop with an Intel Core i7 2.7 GHz processor with 8 GB main memory. The operating system is 64-bit Windows 7.

6.2 Mappings

Mapping CPS: We use real RFID-based baggage tracking data from the transfer system of terminal-3 of Copenhagen Airport (CPH). It has 11 CPS locations with 11 RFID readers deployed. After filtering the erroneous records, there are 2.1M tracking records for 220K distinct bags collected during the period from Dec 21, 2011 to Dec 02, 2013. As most of the locations contain loops, after converting the 2.1M tracking records there are 787K Mapping records. Fig. 5.11a shows the execution time of the CPS mapping algorithm (Algorithm 39) for different numbers of tracking records. It shows that the mapping time scales linearly with the number of tracking records.

Mapping SCPS: We generate 2M tracking records for 300K distinct objects in a floor plan with 50 different symbolic locations. For each object there are

on average 6 tracking records produced. After passing the tracking records into the mapping algorithm (Algorithm 10), there are 2M mapping records. Fig. 5.11b shows the execution time of SCPS mapping algorithm for different numbers of tracking records. It shows that the mapping time scales linearly with the number of tracking records.



(a) Mapping time for CPS **(b)** Mapping time for SCPS

Fig. 5.11: Performance of Mapping algorithms

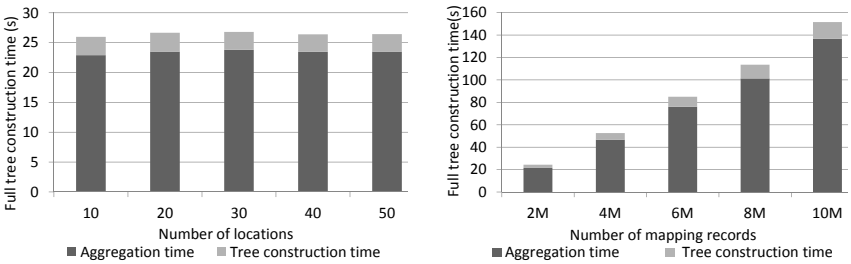
6.3 *DLT*-Index

The page size is set to 4KB and each entry of the *DLT*-Tree is 20 bytes. This yields 204 entries per node. We use both real data as well as synthetic data for experimenting with different aspects of the *DLT*-Index. We use the above mapping table produced from the airport baggage tracking data and scales it to 10M mapping records. For scaling, Max(*ObjectId*) is added to the existing *ObjectId*s for uniqueness, and a random time between 50 to 120 seconds is added to the $time_{start}$ and $time_{end}$ for each record. The semi-real data contains 3M distinct objects and a total of 17.3M distinct time points distributed over the 11 symbolic locations.

While generating synthetic data, On the other hand, we focus on data volume instead of focusing on the real world scenario. We generate 10M mapping records distributed over 50 different symbolic locations. For each location, we generate 200K objects and around 20 objects appearing per minute. In the mapping table there are on average 197K distinct time points created per location and a total of 9.8M distinct time points distributed over 50 locations.

First, the tree construction and update costs are investigated, and then the query processing cost is studied. As part of the investigation of query processing cost, we measure the total number of node accesses with and without pruning. Then we proceed to measure the query processing times of point, interval and other types of queries in our *DLT*-Index, and then compare with the query times using SQL over the RDBMS.

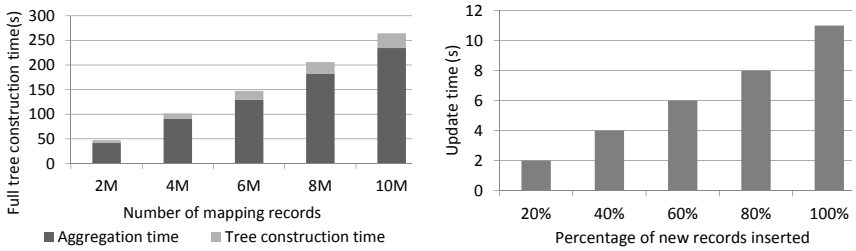
Tree Construction: Before the tree construction the aggregate information C for each time point is precomputed. Then the trees are constructed for the time points including the aggregate information. The full tree construction time including the aggregate information generation time for the synthetic data is reported in Fig. 5.12a and 5.12b. For generating Fig. 5.12a, the total mapping records are kept fixed to 2M. Even though the *DLT*-Index maintains a tree for each location, Fig. 5.12a shows that the number of locations has no effect on the tree construction time. On the other hand, Fig. 5.12b shows that the tree construction time increases linearly with the number of mapping records. In both cases, generating the aggregate information takes approximately 89% of the time during the tree construction as it has to access each record for finding distinct time points, sort and union them to compute the values of c_{total}, c_{enter} and c_{exit} at each time point. Similarly, Fig. 5.13a shows the effect of the number of mapping records on the full tree construction time for the semi-real data. The tree construction takes more time for the semi-real data compared to the synthetic data as the semi-real data contains more distinct time points. Overall, we can see that the tree construction cost depends on the number of records as well as the number of time points but not on the number of locations. The main reason behind this is that the total tree construction time is mainly influenced by the aggregate information generation that in turns depends mainly on the number of records and time points.



(a) Scaling no. of locations **(b)** Scaling no. of mapping records

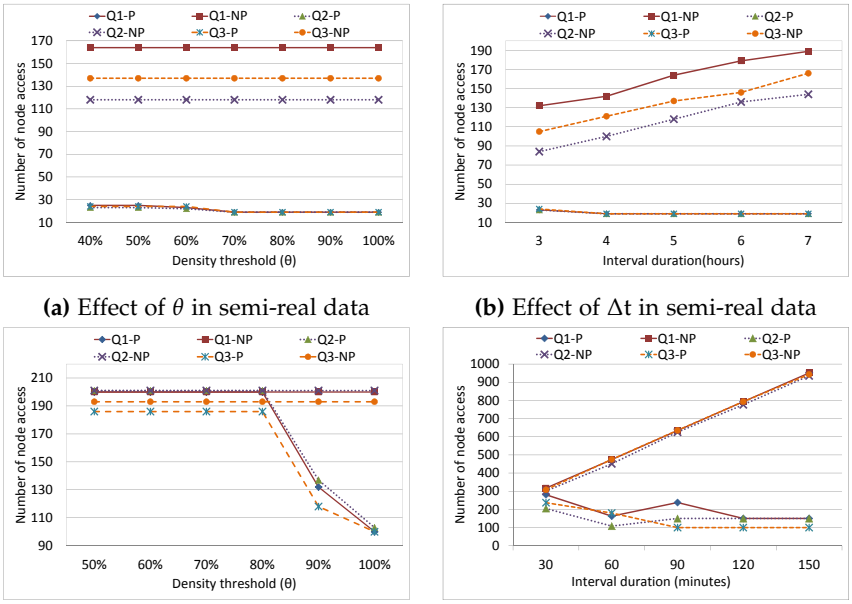
Fig. 5.12: Tree construction cost for synthetic data

Update: We test the update cost of the *DLT*-Index with the semi-real data. First, trees are constructed from 4M mapping records. Then, the trees are updated by inserting new records. The number of inserted records is increased gradually. Fig. 5.13b shows that the update time scales linearly with the number of mapping records. The update cost with the synthetic data is not tested as it is enough to understand the effect from the semi-real data.



(a) Scaling no. mapping records **(b)** Update cost

Fig. 5.13: Tree construction and update cost for semi-real data



(a) Effect of θ in semi-real data **(b)** Effect of Δt in semi-real data

(c) Effect of θ in synthetic data **(d)** Effect of Δt in synthetic data

Fig. 5.14: Processing CBDLQ in the DLT-Index.

6.4 Query Processing

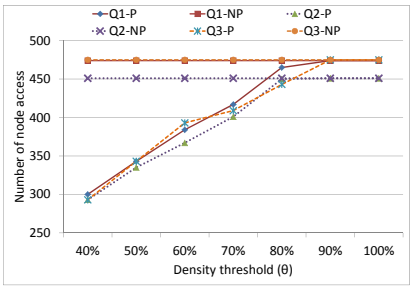
For experimenting with the different aspects of the query processing by the *DLT*-Index, we have generated a set of random queries. For creating the random queries, a built-in function of the RDBMS is used to get random time points from the mapping table. Then we add the necessary durations with the time points to create intervals for the queries.

Three random *CBDLQs* are generated for semi-real data and are processed with pruning (P) and no pruning (NP). The node access for each of them is

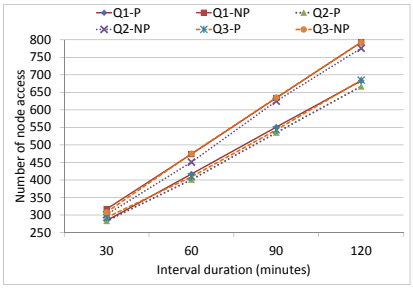
reported in Fig. 5.14a and 5.14b. In all the cases, *P* accesses much fewer nodes compared to *NP*. In Fig. 5.14a the query time interval is kept fixed, and the density threshold(θ) is changed. It shows that, for *NP*, a query has to access the same number of nodes regardless of the value of θ, as it always has to compute the total number of objects for the full interval. However, for *P*, when the value of θ is increased, the number of node accesses decreases as the number of dense locations decreases due to an increase in the value of σ (effect of pruning with *observation 1*). However, the number of node accesses becomes constant after some time. In Fig. 5.14b, θ is kept fixed and the time interval length is changed. It shows that for *NP*, node access increases with the length of the intervals as the query has to cover more time points to complete the interval. However, for *P*, node access decreases and becomes constant at some point. Here, any of the observations (*observation 1* or *observation 2*) can be the reason depending on the number of dense locations. The query is affected by *observation 1* if the number of dense locations decreases and by *observation 2* if the number of dense locations increases.

We also generate three random *CBDLQs* for synthetic data and process with both *P* and *NP*, and the node accesses are reported in Fig. 5.14c and 5.14d. Fig. 5.14c shows a similar effect as shown in Fig. 5.14a. For *P*, the node access is less at the beginning as an effect of the pruning with *observation 2*. The node access also decreases with the increase in θ as the number of dense locations decreases and σ increases (effect of pruning with *observation 1*). Fig. 5.14d shows, for *NP*, the query accesses more nodes with the increase in the duration, whereas the number of node accesses decreases for *P*. It shows that for *P*, the number of node accesses becomes stable at some point.

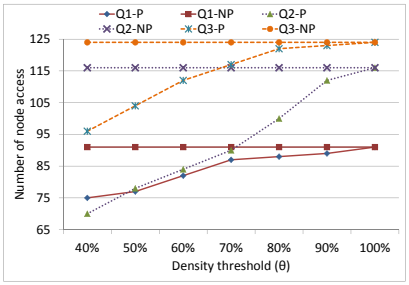
We generate three random *DBDLQs* for synthetic data and process them with both *P* and *NP*. The number of node accesses for each of them is reported in Fig. 5.15a and 5.15b. In all the cases, *P* accesses fewer nodes compared to *NP*. In Fig. 5.15a the query time interval is kept fixed, and the density threshold(θ) is changed. It shows that for *NP*, a query has to access the same number of nodes regardless of the value of θ, as it always has to compute the total duration of objects stays for the complete time interval. However, for *P*, the number of node accesses increases with an increase in the value of θ, as the locations become less dense and *Observation 3* becomes less effective. Moreover, Fig. 5.15a shows that for *P*, the number of node accesses becomes stable when it reaches the total node accesses for *NP*. In Fig. 5.15b, θ is kept fixed and the time interval length is changed. It shows that for both *P* and *NP*, the number of node accesses increases with the interval length as the query has to cover more time points to complete the interval. However, the number of node accesses is always less for *P* than for *NP*. We also generate three random *DBDLQs* for semi-real data and process both with *P* and *NP*. The effects on the node accesses are reported in Fig. 5.15c and 5.15d. In all cases, they show the same effect as for the synthetic data.



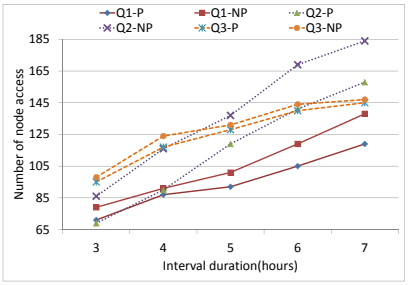
(a) Effect of θ in synthetic data



(b) Effect of Δt in synthetic data



(c) Effect of θ in semi-real data

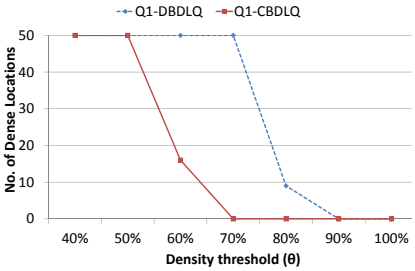


(d) Effect of Δt in semi-real data

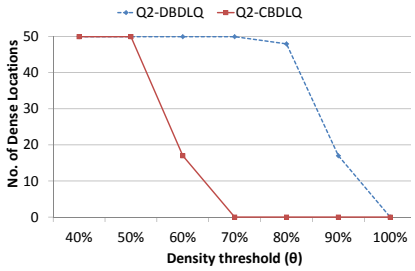
Fig. 5.15: Processing DBDLQ in the DLT-Index.

We generate three random *CBDLQs* and *DBDLQs* for synthetic data and report the number of dense locations for each of them in Fig. 5.16a, 5.16b, and Fig. 5.16c. It shows that, for the lower values of thresholds, the total number of dense locations is the same for both CBDLQ and DBDLQ. However, when the threshold is increased, the DBDLQ always reports more dense locations than CBDLQ. This is also obvious from the definition that, for any location, always DBD $\geq$ CBD as a *CBD* considers only the number of objects whereas a DBD considers both the number of objects and the stay duration of each object.

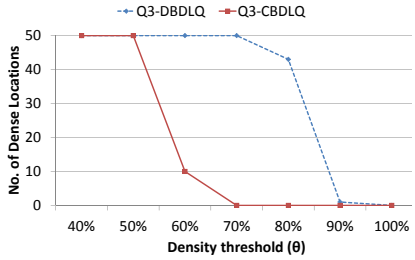
We also process three random queries that find the number of objects at each location for one month interval with a given condition on the aggregate results (like GROUP BY with HAVING condition in SQL) to see the effect of *P* on larger interval in the *DLT*-Index. Fig. 5.16d shows that *P* accesses significantly less number of nodes compared to *NP*.



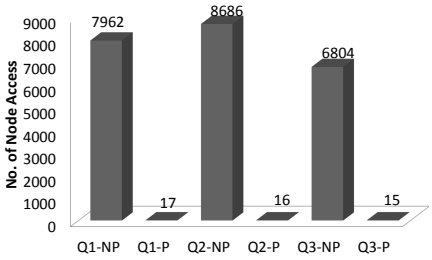
(a) No. of dense locations in CBDLQ and DBDLQ for Q1



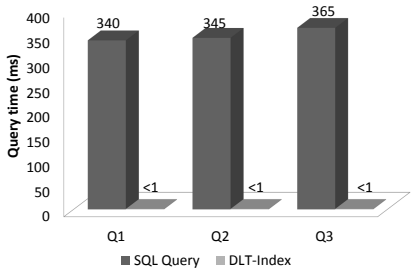
(b) No. of dense locations in CBDLQ and DBDLQ for Q2



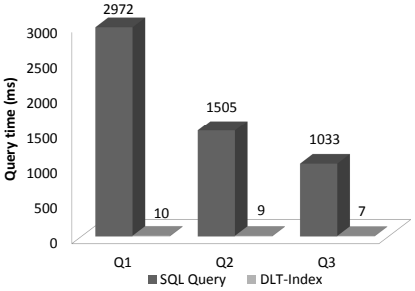
(c) No. of dense locations in CBDLQ and DBDLQ for Q3



(d) Comparing the no. of node accesses in interval query with SQL like GROUP BY clause with HAVING condition in the *DLT*-Index with P and NP



(e) Comparison of *DLT*-Index and RDBMS for point queries



(f) Comparison of DLT-Index and RDBMS for interval queries with SQL like GROUP BY clause

Fig. 5.16: Comparing the number of dense locations in CBDLQ and DBDLQ, pruning effects in processing the interval queries in the *DLT*-Index, and Interval and point query processing times in the *DLT*-Index and SQL in RDBMS.

Listing 5.1: SQL1

```
SELECT location, COUNT (DISTINCT ObjectID)
FROM [mapping table] m
WHERE (m.time_start BETWEEN q_s AND q_e)
        OR (m.time_end BETWEEN q_s AND q_e)
        OR (q_s BETWEEN m.time_start AND m.time_end)
GROUP BY location
HAVING COUNT (DISTINCT ObjectID) > threshold
```

We randomly generate three point queries and also three interval queries that find the number of objects at each location for a given time interval. The interval queries are for one year interval. The queries are processed by both the *DLT*-Index and the relational DBMS. An example interval *SQL* query is shown in Listing 5.1. First, we process the queries inside the DBMS without creating any indices in the *mapping table*. Then we create both clustered and non-clustered indices on different columns and on combination of relevant columns. Further, the SQL queries are processed over the created indices in the RDBMS. However, we find the queries are fastest without any indices in the RDBMS. So, we have reported the query processing durations without without any indices in the RDBMS. For a fair comparison, the first query processing time for each query in the SQL is ignored for warm-up and data loading into main memory. Then we execute each query five times and take the average query time. Fig. 5.16e and 5.16f show that the point query processing in the *DLT*-Index is more than 340 times faster, and in some cases the interval query processing time is around 300 times faster.

7 Related Work

Indoor space modeling for tracking of moving objects has been proposed in [46, 61, 79]. Modeling indoor spaces for the purpose of RFID data cleansing are proposed in [18, 19]. Our ERDG and RDG are inspired by the graph-based indoor space model [46]. That model converts the raw RFID readings into tracking records containing the first and last time when an object appeared within a reader's activation range. In our earlier work [15], we converted the tracking records into stay records containing the transition time between readers. In another earlier work [13], RFID baggage tracking records are converted into *FlightLeg Records* for analyzing baggage status at the airport level. In the present chapter, the tracking records are converted into mapping records showing when an object actually entered and exited the corresponding location.

Much work has been carried out on managing and analyzing the data

from RFID-based indoor and mixed indoor-outdoor tracking applications. Methods for managing and accessing RFID based tracking data have been proposed in [27, 76]. Techniques for storing RFID-based tracking data for analyzing at different levels of abstraction have been proposed in [38, 41]. Methods for finding frequent indoor paths have been proposed in [22, 69]. A number of different kinds of queries for the indoor tracking scenario and their processing techniques have been proposed in [60, 80, 81], where [80] focuses on distance aware join processing, [60] focuses on distance aware query processing, and [81] focuses on nearest neighbor query processing. In the present chapter, we model both constrained and semi-contained indoor movements, provide the algorithms for mappings as well as efficient processing techniques of a new type of indoor query called dense location query.

There are works [40, 45, 52, 55] that address density queries and hot route queries on road networks. However, the geometric space and the distances used in outdoor road networks do not apply to the indoor space and symbolic indoor tracking data in our settings.

There are spatio-temporal data access methods for point and interval queries on temporal dimension in outdoor settings [62, 70, 73, 74] and in indoor settings [47]. These indexing techniques are capable of efficiently accessing individual records for a range of locations as well as for a time period. However, for a DLQ, aggregate information for each location has to be accessed. Our *DLT*-Index itself contains aggregate information and avoids accessing individual data records. There are also works for indexing spatio-temporal data for aggregate queries [58, 64]. The aRB-tree [64] stores the aggregate information in the tree nodes. However, it counts the same objects multiple times if the objects remain in the same location in several timestamps during the query interval. This problem is solved by an approximate approach in [72]. In contrast to the traditional range and point queries where a query generally does not access all the locations, the DLQ has to access all the locations' information to determine whether each location is dense or not. So like the *aRB*-Tree, we also maintain a separate tree for each location. However, we do not use the *R*-Tree part of the *aRB*-Tree which uses MBRs. The distance functions in the symbolic space are different from the Euclidean distance, so the MBRs cannot approximate the distances well. Thus, the MBR used in *aRB*-Tree is not applicable in our case. Although our temporal indexing structure is motivated by [31], we extend it to a symbolic spatio-temporal space and we additionally maintain necessary aggregate information in nodes to facilitate counting distinct objects for a time interval. The aggregate information stored in the non-leaf nodes helps achieve effective pruning in processing DLQs as well as intervals and SQL like GROUP BY queries with HAVING conditions.

8 Conclusion and Future Work

Indoor tracking applications generate a massive volume of tracking data that are at a very low level for further analysis. Preparing such low-level data and use it for finding dense locations can be useful for detecting overloaded areas, crowd and security control, and other location based services. In this chapter, we proposed an approach to extract the dense locations from indoor tracking data. We developed two graph-based models for constrained and semi-constrained indoor movements. The models are useful for mapping the tracking records with the semantic location so that it is possible to know the entry and exit times of an object at a symbolic location. Then we introduced an indexing technique, the Dense Location Time Index (*DLT*-Index), which indexes aggregate information with the time points. We also proposed efficient methods for index construction and updating, processing point, interval, duration, and dense location queries including pruning strategy on the *DLT*-Index. Our experimental evaluation on large amounts of real and synthetic data shows that the *DLT*-Index can process queries efficiently and several orders of magnitude faster than an RDBMS. The *DLT*-Index is also useful for general time interval indexing, which enables efficient processing of queries like asking for the number of distinct records for a specified time point or time interval.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

Chapter 6

Summary of Conclusions and Future Research Directions

Abstract

This chapter summarizes the conclusions and directions for future work presented in Chapters 2-5, Appendices A and B.

1 Summary of Results

The thesis presented different kinds of analytics on indoor moving objects, especially for the airport baggage handling scenario. The Ph.D. study was supported by a major industry initiative called the BagTrack project. The aim of the project is to build a global IT solution to significantly improve the worldwide aviation baggage handling quality. Typical baggage mishandling includes delayed baggage or baggage left behind at the origin airport, missed connecting flight, bag loss, wrong bag destination, etc. The Ph.D. thesis presented a number of data management and mining techniques for efficient analysis of baggage tracking data as well as other general indoor tracking data. In addition to other types of analysis such as indoor dense location extraction, the thesis focused on finding the reasons of baggage mishandling and also predicting risk of baggage in real-time so that the bag can be saved from being mishandled. Further, as analyzing and querying the unstructured raw data is very slow and it is sometimes impossible to get answers in a reasonable time, the thesis took scalability and efficiency into consideration. To solve the efficiency problem as well as facilitating other opportunities, the thesis presented a data warehousing solution and an indexing solution. The developed technologies and concepts were general enough such that they

could be applied to different spatio-temporal data management systems and tracking applications. In the following, each of the presented chapters is summarized with its corresponding important results. Finally, the overall contributions of the PhD thesis are discussed at a higher level of abstraction.

Chapter 2 presented an efficient data warehouse solution for analysis of RFID-based airport baggage tracking data. The presented data warehouse considered relevant and complex dimensions and measures to facilitate complex multidimensional queries and to provide insight into the baggage tracking data as well as finding the reasons of baggage mishandling. In the data warehouse, the huge unprocessed raw RFID tracking data were converted into stay records which resulted in a huge data reduction making it suitable for further analysis, e.g., analyzing the baggage stay durations between locations. The data warehouse introduced the concept of date localization where each date was localized to a particular airport for capturing the special events related to the particular airport. Furthermore, the many-to-many relationship between *Bag* and *Flight* dimensions was handled effectively both in the relational data warehouse and the cube design. The chapter also discussed the necessary steps for the ETL operation and solved the complexities of the ETL operation for loading the data warehouse with the appropriate data from the data source. The proposed data warehouse and cube were implemented and the query processing times in the relational data warehouse, cube, and the source database were compared. The results showed that some queries in the cube were 7 times faster compared to the relational data warehouse and 2300 times faster compared to source data. The chapter also reported some analysis results using a very user friendly BI tool called *Targit BI Suit 2K11* and it showed that the data warehouse can reveal interesting insights in the data set. The proposed data warehouse solution can benefit the aviation industry significantly for storing and analyzing baggage handling data as well as taking actionable decisions for service improvement. The concepts presented in the chapter can be generalized for other kinds of multi-site based symbolic indoor, outdoor, and mixed indoor-outdoor tracking systems.

Chapter 3 presented a detailed methodology for mining risk factors in RFID baggage tracking data. The primary goal was to find interesting patterns and identify risk factors that are related to baggage mishandling. The chapter presented the required pre-processing steps for preparing the low-level RFID tracking data into *FlightLeg records* for a higher level analysis. Several data mining techniques such as *Decision Tree (DT), Naive Bayes classifier (NB), KNN classifier (KNN), Linear regression (LIR), Logistics regression (LOR)*, and *Support vector machine (SVM)* were used for learning predictive models that assigned a probability score called a risk score to each bag representing the probability of being mishandled. The scores were also used for ranking the bags in the data set. The chapter dealt with the class imbalance problem present in the real data set. Based on the AUCs and the precision-recall

curves we found that the *Decision Tree* was the best classifier for the data set. The data set was fragmented into transit, non-transit, shorter and longer transit records and appropriate models for the different fragments were obtained. It was found that re-balancing the data set by under-sampling helped to obtain a better predictive model for the bags with longer transit durations. The chapter reported a comprehensive experimental study and it showed that fragmenting and mining each of the fragments separately was the right choice. The extracted patterns were also reported and they showed that the total available handling time for a bag before its flight is an important factor. They showed that a bag is normally at high risk of being mishandled if it has less than 54 minutes transfer time in the transit airport. Furthermore, a long stay between readers or baggage handling locations and the busyness of the airport, i.e., the number of bags handled during the flight hour are also important factors related to baggage mishandling. The proposed methodology revealed interesting concrete patterns in the data set, thus, the methodology can help the airline industry for examining the problems in baggage management for further improvement in the system.

Chapter 4 presented an online risk prediction system for indoor moving objects. Here, the stay records are used for constructing a probabilistic flow graph (PFG) and the aggregated PFG (APFG). In the graphs, a set of histograms called the least duration probability histogram (LDPH) and aggregated LDPH (ALDPH) is used for capturing the probabilistic information about the object transition times. The risk prediction system uses the graphs for obtaining risk scores for the online moving objects and uses the obtained risk scores to predict the riskiness of the objects. The chapter also presented a cost model for finding the risk score thresholds that maximize the benefit of detecting and removing the predicted risky objects. The presented risk prediction system considered the total available processing time of an object and performed a normalization for specializing the maximum allowable duration of an object for each of its transitions. This normalization and specialization facilitate predicting the risky objects as soon as possible as well as reduce false positives. A comprehensive experiment was conducted with synthetic and real data. The results showed that the proposed risk prediction method can identify the risky objects very accurately when the objects approach the bottleneck locations on their paths. Results showed that APFG with normalization can significantly reduce the operation cost (83.4% in the experimental data set). Further, the experimental results also showed that the prediction system can predict the risky objects well in advance such that they can be saved from being mishandled. The proposed risk prediction system can help the aviation industry for predicting risky bags during their processing at the airport and can significantly reduce the mishandling rate and related operational cost. The system also can be applied to predict the risks of the objects for similar kinds of time critical applications.

Chapter 5 presented an efficient approach for finding dense locations in indoor tracking data. Indoor spaces were categorized into two types, constrained path space (CPS) and semi-constrained path space (SCPS). The chapter presented two graph models: Extended Reader Deployment Graph (ERDG) and Reader Deployment Graph (RDG) to capture the topological information of the CPS and SCPS, respectively. The graphs are used for mapping the tracking records of indoor moving objects into the mapping records representing the entry and exit times of the objects at different symbolic locations. The mapping records can be used for dense location extraction as well as for other kinds of analysis. The chapter introduced a new indexing technique called the Dense Location Time Index (*DLT*-Index), which indexes aggregate information about the objects with the time points from the mapping records. The chapter also presented efficient algorithms for index construction and updating, pruning methods, processing point, interval, duration, and dense location queries through the *DLT*-Index. The chapter reports a comprehensive experimental study with large synthetic and real data. The results showed that the *DLT*-Index can process queries efficiently and the query processing times are several orders of magnitude faster than an RDBMS. Further, the index construction and updating times also showed that the solution is scalable. The *DLT*-Index is also useful for general time interval indexing, which enables efficient processing of queries such as finding the number of distinct records for a specified time point or time interval.

Appendix A and B do not contain any additional content compared to Chapter 5. Appendix A discussed the graph model ERDG for constrained indoor space (CPS). It also presented the mapping techniques for the objects moving in a CPS. Appendix A also discussed the naive approach of processing a dense location query for the CPS scenario. Appendix B extended Appendix A by adding graph model RDG for semi-constrained indoor space (SCPS), mapping techniques for the moving objects in an SCPS, the *DLT*-Index, and dense location query processing techniques. Chapter 5 extended Appendix A and B significantly by adding new density definition, mapping algorithms, and algorithms for index construction, updating, and different kind of query processing. Additionally, Chapter 5 presented an experimental study showing several aspects of the proposed methods.

In summary, indoor spaces touch several aspects of our lives. The tracking data produced by the indoor tracking systems are enormous and can be very useful for decision making, location-based services, problem finding, and system improvement. However, unprocessed huge raw indoor tracking data are nontraditional and complex in nature, thus, making it difficult for further analysis. The thesis proposed a number of data management techniques for efficient and effective analysis of indoor tracking data. The Ph.D. study focused on airport baggage tracking scenario, as studies showed that the aviation industry lost more than 3,300M USD/year and they face a

major challenge to solve the problem of mishandled bag. Although the baggage tracking scenario inherits the general problems of indoor tracking, it becomes more unique in nature due to the constrained movements of baggage in conveyor belts and the locations and times are much more fine-grained. Due to the complex nature and high volume, the query processing on the RFID baggage tracking data was very slow and very often impossible to get more insight into the data. The presented data warehouse in Chapter 2 took all the complexities into consideration and solved the problems. It enables multidimensional data analysis for gaining more insight into the RFID baggage tracking data and made it suitable for a nontechnical user such that they can access, visualize, and analyze the data very easily. Overall, the data warehouse facilitates complex and advanced analytical query processing very efficiently over the baggage tracking data. Although the proposed data warehouse offers manual data exploration, however, cannot find the hidden patterns that are co-related to baggage mishandling. The presented data mining methodology in Chapter 3 facilitates automatic extraction of interesting patterns and factors that are closely related to baggage mishandling. The methodology also discussed how to deal with a class imbalanced data. Overall, the presented data mining methodology can help the aviation industry by finding interesting patterns and factors that are co-related to baggage mishandling. However, the presented data mining methodology considered the analysis of the tracking data in an offline scenario. Chapter 4 offers an online risk prediction (ORP) system that can facilitates real-time notifications about the online risky indoor moving objects, e.g., notifying risky bags during their operation in the airport. It features immediate actions to the predicted risky bags such that they can be saved from being mishandled. As a result, the ORP system can considerably reduce the baggage handling problem and reduce the cost of baggage mishandling significantly. In Chapter 5, the thesis presented an efficient approach for finding dense locations in indoor tracking data as it can facilitates finding overloaded indoor locations which can help to change the infrastructure, balancing loads, crowd management, and other location based services. The chapter defined three different kinds of indoor dense locations and two different graph models which covers various aspects of indoor settings. Moreover, the proposed indexing technique supports several kinds of queries very efficiently. Overall, the proposed methods can extract indoor dense locations from large indoor tracking data efficiently. In brief, the thesis has presented several aspects of analytics on indoor moving objects and special focus was given to the airport baggage handling scenario. The thesis resolved the many complexities in RFID baggage tracking data by different efficient data management techniques and made it suitable for advanced analysis and obtaining more insight into the data. A number of novel data management techniques were proposed that can significantly contribute to the aviation industry as well as to the spatio-temporal data management

and data mining research.

2 Future Research Directions

Several directions for future work exist for the methods and techniques presented in this thesis. In the following, future research directions for each of the presented chapters are summarized.

There is a number of future advancements on the data warehouse concepts presented in Chapter 2. New pre-aggregation strategies are required to scale the proposed data warehouse solution for thousands of airports maintaining more precise bag movements over long time periods. Considering the complex movements of baggage from the origin to the final destination, new research on building data cubes on spatio-temporal data with complex topologies is an interesting future work. More efficient representations of *spatio-temporal sequences*, e.g., flight sequences, route sequences, etc., in the data warehouse will be interesting for query processing on such sequences. Efficient indexing structures also should be designed on such sequences as well as other spatio-temporal dimensions. Moreover, in order to get both more seamless querying and better performance, native support for spatio-temporal sequences within DW and BI tools should be developed. Technologies on real time data warehousing and ETL operation also need to be studied in the baggage tracking or similar types of context.

There are several directions for future work on the data mining methodology presented in Chapter 3. There should be a more thorough study for mining the non-trivial root causes for baggage mishandling, given the low Mishandled rate. Another work will be analyzing baggage handling sequences for finding bottleneck in the system. Additionally, finding spatio-temporal outliers from the RFID baggage tracking data will be an interesting work. Furthermore, developing native support from the data mining tools like automatic methods for finding the most appropriate models will be another future work.

There is a number of future advancements on the risk prediction system presented in Chapter 4. The presented system formulated for constrained indoor movements. In the future, the proposed techniques should be expanded for capturing more general scenarios such as semi-constrained indoor movements and mixed indoor-outdoor tracking. In addition to that, predicting risks for the objects in a scenario where the paths of the objects are not known in advance will be another future direction. It is also interesting to see the behavior of the risk prediction models when adding more dimensions such as weekday, daytime, etc.

In the case of Chapter 5, mining dense or busy time periods for different locations in a day or different week days will be an interesting future research

direction. Further, continuous query processing for dense location detection in online indoor tracking data will be another research direction.

Overall, the thesis addressed several issues in indoor tracking data management, especially in the baggage tracking scenario. However, there can be a variety of future interesting research topics. In future, we will continue and explore this field further by conducting research on a wide range of topics related to indoor moving objects. First, we will explore the baggage handling scenario to propose data management solutions and build predictive models to resolve the transfer baggage problems. Second, RFID baggage tracking data can become very large if they are collected from thousands of airports for a long time period. So, it will be very challenging to access, manage, process, and use the big RFID baggage handling data for analysis. In future, we will explore the challenges of big RFID baggage handling data analytics. Third, RFID and similar kinds of indoor tracking systems are error-prone by nature. There is research that addresses data cleansing methods for indoor RFID settings [18–20]. In future, we will integrate data cleansing methods with the proposed solutions. Fourth, we will perform analytics on the big tracking data coming from moving objects in indoor and mixed indoor-outdoor spaces. Moreover, we will also consider other, more general indoor and mixed indoor-outdoor tracking scenarios for social relation analysis. It will be interesting to perform multidimensional sequential pattern mining for finding typical travel patterns in large indoor spaces that can help to optimize operations and changes in the infrastructure. Moreover, we will also study the tracking data for finding interesting patterns that can create new business opportunities and new location based services. Moreover, enabling more useful operational queries in the tracking data, finding efficient routes or shortest path with given constraints, etc., will be other future directions. In addition to the offline data analysis, focusing on real-time or online applications will be more interesting for future advancement.

References

[1] The BagTrack project. http://daisy.aau.dk/bagtrack. Accessed: 2015-08-12.

[2] BIDS Helper. http://bidshelper.codeplex.com/. Accessed: 2015-08-12.

[3] KNIME analytics platform. https://www.knime.org/knime. Accessed: 2016-02-27.

[4] Linear regression. https://en.wikipedia.org/wiki/Linear_regression. Accessed: 2016-02-27.

[5] Logistic regression. https://en.wikipedia.org/wiki/Logistic_regression. Accessed: 2016-02-27.

[6] Lyngsoe Systems. http://www.lyngsoesystems.com/.

[7] MDX Studio. www.sqlbi.com/tools/mdx-studio/. Accessed: 2015-08-12.

[8] Schengen area. http://www.schengenvisainfo.com/schengen-visa-countries-list/. Accessed: 2016-02-27.

[9] SITA. www.sita.aero.

[10] SITA baggage report 2014. http://www.sita.aero/resources/type/infographics/baggage-report-2014. Accessed: 2015-08-12.

[11] TARGIT. www.targit.com/.

[12] Venture Development Corporation (VDC). http://www.vdc-corp.com/.

[13] T. Ahmed, T. Calders, and T. B. Pedersen. Mining risk factors in RFID baggage tracking data. In *MDM (1)*, pages 235–242, 2015.

[14] T. Ahmed, T. B. Pedersen, and H. Lu. Capturing hotspots for constrained indoor movement. In *SIGSPATIAL/GIS*, pages 462–465, 2013.

[15] T. Ahmed, T. B. Pedersen, and H. Lu. A data warehouse solution for analyzing RFID-based baggage tracking data. In *MDM (1)*, pages 283–292, 2013.

[16] T. Ahmed, T. B. Pedersen, and H. Lu. Finding dense locations in indoor tracking data. In *MDM (1)*, pages 189–194, 2014.

[17] R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. In *ECML*, pages 39–50, 2004.

[18] A. I. Baba, H. Lu, T. B. Pedersen, and X. Xie. A graph model for false negative handling in indoor RFID tracking data. In *SIGSPATIAL/GIS*, pages 454–457, 2013.

[19] A. I. Baba, H. Lu, T. B. Pedersen, and X. Xie. Handling false negatives in indoor RFID data. In *MDM*, pages 117–126, 2014.

[20] A. I. Baba, H. Lu, X. Xie, and T. B. Pedersen. Spatiotemporal data cleansing for indoor RFID tracking data. In *MDM*, pages 187–196, 2013.

[21] A. Baniukevic, D. Sabonis, C. S. Jensen, and H. Lu. Improving Wi-Fi based indoor positioning using bluetooth add-ons. In *MDM*, volume 1, pages 246–255, June 2011.

[22] Z. Berenyi and H. Charaf. Retrieving frequent walks from tracking data in RFID-equipped warehouses. In *HSI*, pages 663 –667, 2008.

[23] Z. Berenyi and H. Charaf. Utilizing tracking data in rfid-equipped warehouses. In *ICC*, pages 169–173, May 2008.

[24] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

[25] M. A. Bramer. *Principles of Data Mining, Second Edition*. Undergraduate Topics in Computer Science. Springer, 2013.

[26] H. Cao, N. Mamoulis, and D. W. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, pages 82–89, 2005.

[27] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. E. Sarma. Managing RFID data. In *VLDB*, pages 1189–1195, 2004.

[28] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 2002.

[29] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *ICML*, pages 233–240, 2006.

[30] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *SIGKDD*, pages 155–164, 1999.

[31] R. Elmasri, G. T. J. Wuu, and Y.-J. Kim. The time index: An access structure for temporal data. In *VLDB*, pages 1–12, 1990.

[32] B. Fazzinga, S. Flesca, E. Masciari, and F. Furfaro. Efficient and effective RFID data warehousing. In *IDEAS*, pages 251–258, 2009.

[33] S. Feldmann, K. Kyamakya, A. Zapater, and Z. Lue. An indoor Bluetooth-based positioning system: Concept, implementation and experimental evaluation. In *International Conference on Wireless Networks*, pages 109–113, 2003.

[34] S. Ferdous, L. Fegaras, and F. Makedon. Applying data warehousing technique in pervasive assistive environment. In *PETRA*, 2010.

[35] G. Gidófalvi and T. B. Pedersen. Spatio-temporal rule mining: Issues and techniques. In *DaWaK*, pages 275–284, 2005.

[36] H. Gonzalez, J. Han, and X. Li. Flowcube: Constructuing RFID flowcubes for multi-dimensional analysis of commodity flows. In *VLDB*, pages 834–845, 2006.

[37] H. Gonzalez, J. Han, and X. Li. Mining compressed commodity workflows from massive RFID data sets. In *CIKM*, pages 162–171, 2006.

[38] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive RFID data sets. In *ICDE*, page 83, 2006.

[39] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems (TODS)*, 25(1):1–42, 2000.

[40] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-line discovery of dense areas in spatio-temporal databases. In *SSTD*, pages 306–324, 2003.

[41] J. Han, H. Gonzalez, X. Li, and D. Klabjan. Warehousing and mining massive RFID data sets. In *ADMA*, pages 1–18, 2006.

[42] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[43] J. Huang and C. X. Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Trans. Knowl. Data Eng.*, 17(3):299–310, 2005.

[44] Y. Huang, L. Zhang, and P. Zhang. A framework for mining sequential patterns from spatio-temporal event data sets. *IEEE TKDE*, 20(4):433–448, 2008.

[45] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective density queries on continuouslymoving objects. In *ICDE*, page 71, 2006.

[46] C. S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, pages 122–131, 2009.

[47] C. S. Jensen, H. Lu, and B. Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, pages 208–227, 2009.

[48] C. S. Jensen, H. Lu, and B. Yang. Indoor - A new data management frontier. *IEEE Data Eng. Bull.*, 33(2):12–17, 2010.

[49] C. S. Jensen, T. B. Pedersen, and C. Thomsen. *Multidimensional Databases and Data Warehousing*, chapter 3, page 40. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[50] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker. *The Data Warehouse Lifecycle Toolkit*. John Wiley and Sons, 2 edition, 2008.

[51] N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, W. H. Engelmann, et al. The national human activity pattern survey (NHAPS): a resource for assessing exposure to environmental pollutants. *Journal of exposure analysis and environmental epidemiology*, 11(3):231–252, 2001.

[52] C. Lai, L. Wang, J. Chen, X. Meng, and K. Zeitouni. Effective density queries for moving objects in road networks. In *WAIM*, pages 200–211, 2007.

[53] C.-H. Lee and C.-W. Chung. Efficient storage scheme and query processing for supply chain management using rfid. In *SIGMOD Conference*, pages 291–302, 2008.

[54] C.-H. Lee and C.-W. Chung. Rfid data processing in supply chain management using a path encoding scheme. *IEEE Trans. Knowl. Data Eng.*, 23(5):742–758, 2011.

[55] X. Li, J. Han, J.-G. Lee, and H. Gonzalez. Traffic density-based discovery of hot routes in road networks. In *SSTD*, pages 441–459, 2007.

[56] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory undersampling for class-imbalance learning. pages 539–550, 2009.

[57] Y. Liu, Y. Zhao, L. Chen, J. Pei, and J. Han. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. *IEEE Trans. Parallel Distrib. Syst.*, 23(11):2138–2149, 2012.

[58] I. F. V. López, R. T. Snodgrass, and B. Moon. Spatiotemporal aggregate computation: a survey. *TKDE*, 17(2):271–286, 2005.

[59] V. López, A. Fernández, S. García, V. Palade, and F. Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Inf. Sci.*, 250:113–141, 2013.

[60] H. Lu, X. Cao, and C. S. Jensen. A foundation for efficient indoor distance-aware query processing. In *ICDE*, pages 438–449, 2012.

[61] H. Lu, B. Yang, and C. S. Jensen. Spatio-temporal joins on symbolic indoor tracking data. In *ICDE*, pages 816–827, 2011.

[62] M. A. Nascimento, J. R. O. Silva, and Y. Theodoridis. Evaluation of access structures for discretely moving points. In *SSTD*, pages 171–188, 1999.

[63] P. Nielsen and U. Parui. *Microsoft SQL Server 2008 Bible*, chapter 11. John Wiley and Sons, 2011.

[64] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang. Indexing spatio-temporal data warehouses. In *ICDE*, pages 166–175, 2002.

[65] T. B. Pedersen. Dimension. In *Encyclopedia of Database Systems*, pages 836–836. Springer, 2009.

[66] D. Pfoser, C. S. Jensen, Y. Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *Proceedings of VLDB*, pages 395–406, 2000.

[67] J. C. Platt. 12 fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.

[68] F. J. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *KDD*, pages 43–48, 1997.

[69] L. Radaelli, D. Sabonis, H. Lu, and C. S. Jensen. Identifying typical movements among indoor objects - concepts and empirical study. In *MDM (1)*, pages 197–206, 2013.

[70] M. Romero, N. R. Brisaboa, and M. A. Rodríguez. The smo-index: a succinct moving object structure for timestamp and interval queries. In *SIGSPATIAL/GIS*, pages 498–501, 2012.

[71] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *VLDB*, pages 544–555. Citeseer, 1996.

[72] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggregation using sketches. In *ICDE*, pages 214–225, 2004.

[73] Y. Tao and D. Papadias. Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In *VLDB*, pages 431–440, 2001.

[74] Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis. Spatio-temporal indexing for large multimedia applications. In *ICMCS*, pages 441–448, 1996.

[75] N. Viswanadham, A. Prakasam, and R. Gaonkar. Decision support system for exception management in rfid enabled airline baggage handling process. In *CASE*, pages 351 –356, 2006.

[76] F. Wang and P. Liu. Temporal management of RFID data. In *VLDB*, pages 1128–1139, 2005.

[77] R. Want. *RFID Explained: A Primer on Radio Frequency Identification Technologies*. Morgan & Claypool, 2006.

[78] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 111–122. IEEE, 1998.

[79] M. F. Worboys. Modeling indoor space. In *ISA*, pages 1–6, 2011.

[80] X. Xie, H. Lu, and T. B. Pedersen. Distance-aware join for indoor moving objects. *TKDE*, 27(2):428–442, 2015.

[81] B. Yang, H. Lu, and C. S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, pages 335–346, 2010.

[82] H. Zhang and J. Su. Naive bayesian classifiers for ranking. In *ECML*, pages 501–512, 2004.

Paper A

Capturing Hotspots for Constrained Indoor Movement

The paper has been published in the
Proceedings of the 21st ACM SIGSPATIAL GIS, Orlando, Florida, USA, pp. 472–475, 2013. The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1145/2525314.2525463

Abstract

Finding the hotspots in large indoor spaces is very important for getting overloaded locations, security, crowd management, indoor navigation and guidance. The tracking data coming from indoor tracking are huge in volume and not readily available for finding hotspots. This chapter presents a graph-based model for constrained indoor movement that can map the tracking records into mapping records which represent the entry and exit times of an object in a particular location. Then it discusses the hotspots extraction technique from the mapping records.

1 Introduction

Technologies like RFID, Bluetooth, etc., enable a variety of indoor tracking applications like people's movement tracking in large indoor space (e.g., airport, shopping mall, museum, etc.), airport baggage tracking, items movement tracking in supply chain system, etc. The huge amount of tracking data generated by these types of systems is very useful for analyzing and decision making. Detection of hotspots in an indoor space like airport baggage tracking will help the authority to manage the overloaded locations of the baggage handling and handle the bags efficiently. In case of airport people movement, detection of hotspots will give the idea about where and when most of the peoples generally gather that can help the authority to manage the crowd and for business it can be a good idea for different location-based services.

It is unsuitable for indoor trajectories to use the geometric polyline representation that is used for outdoor trajectories. For example if an object moves from one room to another then we will get two consecutive tracking records which represent the object location in different rooms. But due to the drawback of indoor positioning technologies, the locations between these two records are not obtained. As a result, it is not easily available when an object enters and exits a particular location. Thus, it is also not easily available how dense a location is. We take all of these complexities into consideration and propose an approach for extracting hotspots from indoor tracking data. To the best of our knowledge, this is the first chapter to consider how to capture hotspots from indoor tracking data with constrained object movement.

Indoor space modeling for tracking of moving objects has been proposed in [46, 79]. We propose a graph based model which is highly motivated by the model proposed in [46]. Their model converts the raw RFID readings into tracking records containing the first and last time of an object appeared within a reader's activation range. In our previous work [15], we converted the tracking records into stay records containing the transition time between readers. In the present chapter, the tracking records are converted into mapping records showing when an object actually entered and exited the corresponding location. There are many works available for online density queries and hot route queries on road networks [40, 45, 55]. However, the scenario of symbolic indoor tracking is different from outdoor tracking as the geometric position of the object is not available in the indoor setting.

The remainder of the chapter is organized as follows. Section 2 discusses the problem formulation. Section 3 describes the mapping of tracking records for semantic locations with graph-based model. Section 4 presents the hotspot queries. Finally, Section 5 concludes the chapter and discusses possible future research.

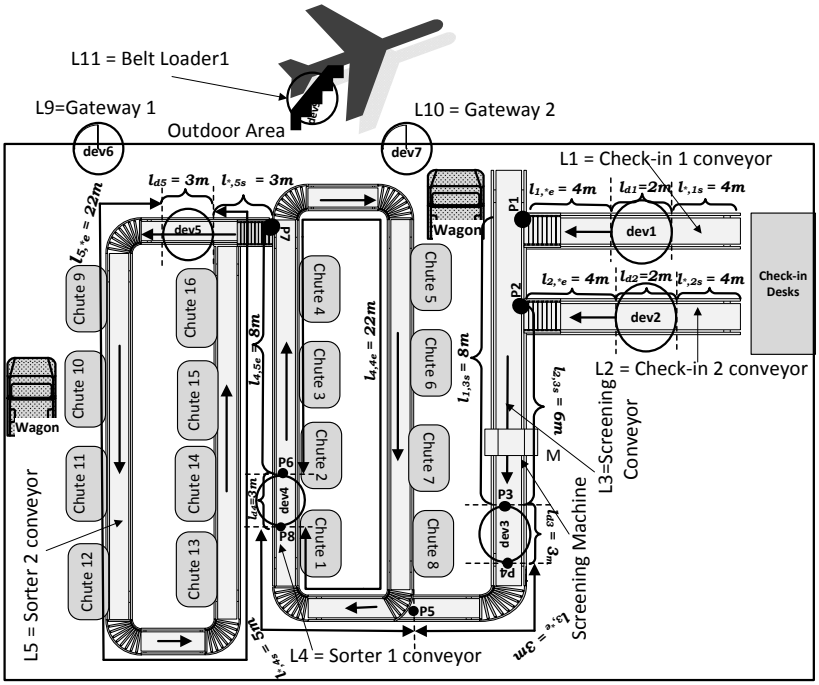


Fig. A.1: Constrained Path in airport baggage management

2 Problem Formulation

Problem Scenario. We assume a setting where the paths between the locations are constrained and objects are continuously moving from one location to another. We call such location as constrained path (CP) symbolic location. The objects cannot move freely and the locations are in some sense one dimensional. The size of a CP symbolic location is measured by length not by area. Fig. A.1 shows an example of a CP, which is a conveyor of an airport baggage handling system. The conveyor is divided into different symbolic locations like check-in 1, check-in 2, screening, sorter-1, sorter-2 and chutes. More detail about the baggage tracking process can be found in [15]. In our setting, the tracking devices are strategically deployed at different fixed locations inside the indoor space, e.g., each section of conveyor belts. The objects contain tags or devices which can be tracked by the tracking devices. For example, in case of RFID technology, the tracking devices are RFID readers and the objects contain RFID tags. Different tracking devices have different sensing ranges. After deployment of the tracking devices, their positions are recorded in the database. In Fig. A.1 the circles represent the deployment of the RFID readers and their tracking ranges. When an object comes un-

der a tracking device's activation range, it is continuously detected by the tracking device with a sampling rate and it generates raw reading records with the form: *(trackingDeviceID, ObjectID, t)*. It means that a tracking device *trackingDeviceID* detects a moving object *ObjectID* in its activation range at timestamp t. A *TrackingRecord(recordID, ObjectID, TrackingDeviceID,* t_{in}, t_{out}*)* table [15] is constructed from the raw tracking sequence, where *recordID* is tracking record identifier and t_{in}, t_{out} respectively represent the timestamps of first reading and last reading of *ObjectID* by *TrackingDeviceID* in its activation range. An example of a table containing tracking records of an object $o1$ from Fig. A.1 is shown in Table A.1. In this table the record rec_1 means that object o_1 is observed by tracking device dev_1 from time 4 to 5, and record rec_3 means that o_1 is observed by dev_3 from time 15 to 18. Due to the limitation of indoor positioning systems, it is unknown what position of o_1 is between 6 and 14 without knowing the floor plan.

Table A.1: Tracking Records of Indoor Moving Objects

RecordID	ObjectID	TrackingDeviceID	t_{in}	t_{out}
rec1	*o1*	*dev1*	*4*	*5*
rec3	*o1*	*dev3*	*15*	*18*
rec5	*o1*	*dev4*	*26*	*29*
rec8	*o1*	*dev4*	*51*	*54*
...	...	...	...	...

Problem Definition. Let L be the set of all symbolic locations inside a large indoor space, $L = \{l_1, l_2, l_3, \ldots, l_k\}$. The *capacity* of location l_i is denoted by c_i = *capacity*(l_i). The *capacity* of a CP symbolic location is a function of *length*. For example, the *capacity* of *check-in 1* conveyor in Fig. A.1 depends on its length.

Definition 1 (Capacity). The *capacity* of a location l_i is the numbers of objects that can be reside at l_i during a defined time unit.

For example, the *capacity* of *check-in 1* conveyor in Fig. A.1 can be 20 objects per minute.

Definition 2 (Density). Let n_i be the number of distinct objects at location l_i during the time interval, $w = [t_{start}, t_{end}]$ and c_i = capacity(l_i) be the capacity of location l_i. Then *density* of location l_i for interval w is defined as,

$d_i = \frac{n_i}{\Delta t \times capacity(l_i)} \times 100\%$, where $\Delta t = t_{end} - t_{start}$.

From the definition we can see that, the value of *density* gives us how dense a location is as a percentage value.

Definition 3 (Hotspot). A location l_i can be considered as a *hotspot* for interval w if d_i exceeds a given threshold θ.

Definition 4 (Hotspot Query). Find all the *hotspots* $H \subseteq L$, for time interval w.

3 Semantic Location Mappings

A tracking device covers a very small portion of a location. As a result it is not sufficient to know when an object actually entered ($time_{start}$) and exited ($time_{end}$) the corresponding location. So there must be a mapping strategy for retrieving such location and timing information.

Modeling Symbolic Locations. In our setting each symbolic location contains only one tracking device deployed in it. For example in Fig. A.1 *check-in 1* is represented by *dev1*. After passing *dev1* and *dev3* when a bag goes to *sorter-1* it will be read by *dev4* and then it may go to *sorter-2* or *chute* or it may circulate within *sorter-1*. For mapping between tracking records and the semantic locations, a reader deployment graph (RDG) can be constructed from the indoor plan given in Fig A.1. Relevant details about the concept of reader deployment graph can be found elsewhere [46]. Although an RDG is capable of mapping the location of an object from the tracking records, it does not provide sufficient information for mapping the *tracking* entry and exit time to the *actual* entry and exit time. For more precise entry time and exit time we extend the RDG with a more detailed model called the Extended Reader Deployment Graph (ERDG). For this, some definitions are needed:

Definition 5 (Covered distance). Given a path p and a tracking device d, the Covered distance (CD) is the length of the part of p that is covered by d's detection range. CD for a tracking device dev_i is denoted as l_{di}. For example in Fig. A.1 $l_{d1} = 2m$ shows the CD of $dev1$ at $L1$.

Definition 6 (Entry lag distance). The *entry lag distance (ENLD)* from location L_x to L_y denoted as l_{x,y_s} is the distance from the ending point of L_x to the first reading point at L_y.

For example, consider Fig. A.1. The journey of an object at location $L3$ can start from either points $P1$ or $P2$ depending on whether the object is coming from $L1$ or $L2$. While moving at $L3$ the object will be first tracked by *dev3* when it comes at point $P3$. Here the distance between the point $P1$ and $P3$ is the *Entry lag distance (ENLD)* which is denoted as $l_{1,3s}$ and similarly ENLD between $P2$ and $P3$ is denoted as $l_{2,3s}$. It can be seen that a location L_y can have many ENLDs depending how many locations end at L_y. In our running example $l_{1,3s} = 8m$ and $l_{2,3s} = 6m$. However we use a special notation $l_{*,ys}$, which indicates that the ENLD at L_y is same regardless of where an object is coming from. In our example $l_{*,4s} = 5m$ is the ENLD of location $L4$ from any location ended at $L4$.

Definition 7 (Exit lag distance). Conversely the *exit lag distance (EXLD)* from location L_x to L_y denoted as l_{x,y_e} is the distance from the last reading point at L_x to the exit point of L_x that leads to location L_y.

Similar to ENLD, let us consider Fig. A.1. The journey of an object at location $L3$ ends when it passes the point $P5$ and reaches location $L4$. While

traveling through $L3$ the object was last detected by $dev3$ when it was at point $P4$. Here the distance between P4 and P5 is the *Exit lag distance (EXLD)* of $L3$ which is denoted as $l_{3,4e}$. As $L3$ has only one destination, the EXLD of $L3$ is always same regardless of destination. So instead of using $l_{3,4e}$ we use $l_{3,*e}$ in this case. In our example the value of $l_{3,*e}$ is 3 meters. Similar to ENLD, a location can have many EXLDs. For example an object can leave location $L4$ by going to $L5$ through $P7$ or can circulate in $L4$ and leave within any point between $P6$ and $P8$. As a result $L4$ has two EXLDs $l_{4,5e} = 8m$ and $l_{4,4e} = 22m$.

The ERDG is formally defined by a labeled directed graph $G = (L, E, T, lb_E)$:

1. L is the set of locations where each location is represented as a vertex in the graph. If a location does not contain any tracking device deployed in it then the corresponding location is labeled as a *virtual location* L_{v_x} where x is an integer.

2. E is the set of directed edges: $E = \{(l_i, l_j) \mid l_i, l_j \in L\}$.

3. T is a set of tuples of the form $\langle$D, Flag, L_{dx}, $\{L_s\}$, $\{L_e\}$ $\rangle$, where D is a tracking device, Flag indicates whether it is a CP or not, $\{L_{dx}\}$ is the CD of D, $\{L_s\}$ is a collection of ENLDs and L_e is a collection of EXLDs.

4. lb_E is a function lb_E: $E \rightarrow T$ that labels an edge by a tuple from T. An edge $(l_i, l_j) \in E$ is labeled by a tuple $T_{i,j}\langle d_k, l_{d_k}, l_{i,js}, l_{j,*e}\rangle \in T$ where d_k is a tracking device deployed at location l_j, l_{d_k} is the CD for d_k, $l_{i,js}$ is an ENLD and EXLD for all the out-going locations from l_j is shown as $l_{j,*e}$. An edge is labeled by a tuple $T_{v_x,j} \in T$ if virtual tracking device dev_{v_x} is assumed to be deployed at location L_{v_x}. Fig. A.2 shows an example of the ERDG of the floor plan of Fig. A.1. Let us consider *edge (L1,L3)* where the tuple $T_{1,3}$ is assigned. The content of the tuple is the tracking device $dev3$ which is deployed at $L3$, l_{d3} which is CD for $dev3$, $l_{1,3s}$ is the ENLD from $L1$ to $L3$ and $l3,*e$ is the EXLD from $L3$ to any next destination.

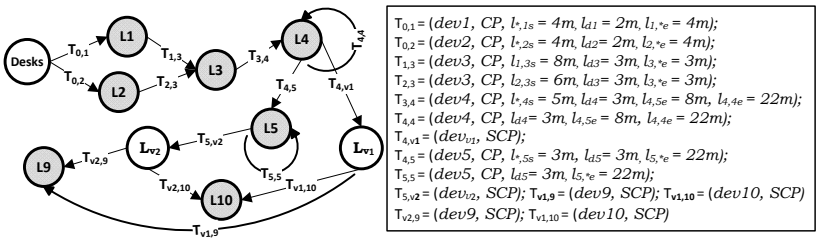


Fig. A.2: Extended Reader Deployment Graph (ERDG)

We define three mapping structures: location to device-In $L2DIn$: L $\rightarrow D$, location to device-Out $L2DOut$: L $\rightarrow 2^D$ and Device to Location $D2L$: D $\rightarrow L$, where L is the set of all locations and D is the set of all tracking devices. For a location l, $L2DIn(l)$ returns the tracking device deployed at l. From the graph it returns the tracking device which is labeled in any edge(s) where l is the destination. Since a CP location contains only one tracking device, all the incoming edges of a location will be labeled by same tracking device. On the other hand $L2DOut(l)$ returns all the tracking device(s) which are labeled in edge(s) where l is the source. These devices are deployed in the adjacent next locations of l. In the third mapping for a tracking device dev, $D2L(dev)$ returns the location of dev, that means the destination vertex of the edge that has dev in its label. In the running example of Fig. A.2 $L2DOut(L4)$ = {$dev4$, $dev5$, dev_{v1}}, $L2DIn(L4)$ = $dev4$, and $D2L(dev4)$ = $L4$.

Mapping for CP Symbolic Locations. For mapping the $time_{in}$ and $time_{out}$ of an object o at a tracking device dev into the $time_{start}$ and $time_{end}$ of o at location l we use the topological information described in the ERDG in Fig. A.2. However both of these values depend on the speed of o at l. We use Eq. (A.1), (A.2) and (A.3) for deriving the $speed$, $time_{start}$ and $time_{end}$ respectively. In all these equations $time_{in}$ and $time_{out}$ are taken from tracking records at $L2DIn(l)$. In Eq. (A.1) $CD\ (dev_x)$ represents the CD of $L2DIn(l) = dev_x$. In Eq. (A.2) the $ENLD$ depends on where the object is coming from and the value is taken from tuple $T_{prevLoc,l}$. In Eq. (A.3) the $EXLD$ is taken from tuple $T_{l,nextLoc}$.

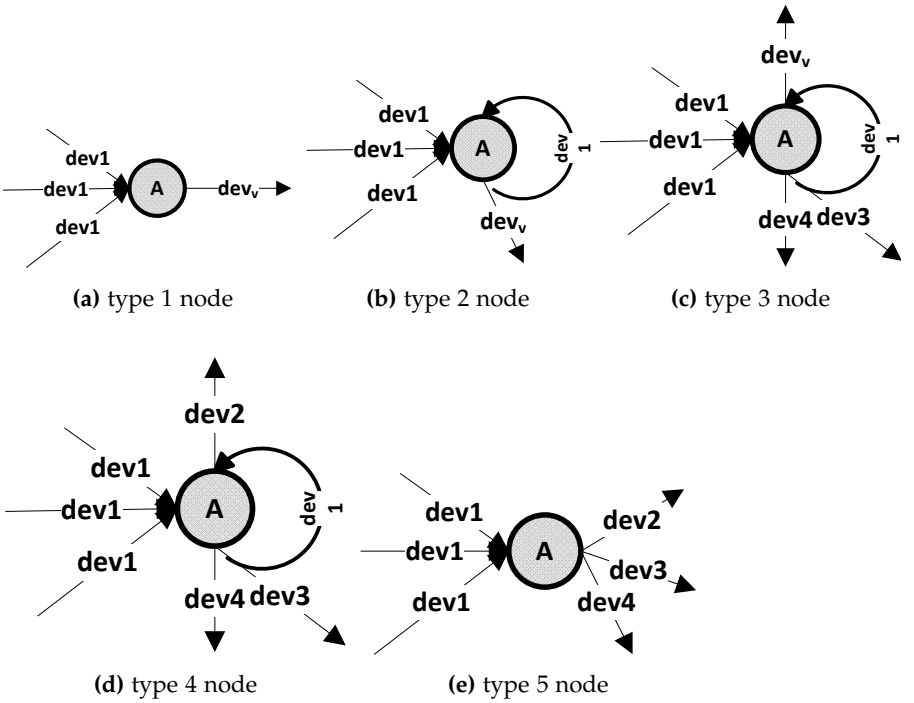
$$Speed := \frac{CD(dev_x)}{(time_{out} - time_{in})} \tag{A.1}$$

$$time_{start} := time_{in} - \frac{ENLD}{Speed} \tag{A.2}$$

$$time_{end} := time_{out} + \frac{EXLD}{Speed} \tag{A.3}$$

For example, consider the second tracking record $\langle o_1, dev_3, 15, 18\rangle$ of Table A.1. From the graph, CD of $dev3$ = 3 meters and $D2L(dev3)$ = L3. So the $speed$ of o_1 at location $L3$ is : $speed = \frac{3}{18-15}$ = 1 meter/second (we assume the duration is measured in seconds). Similarly we can find the $time_{start}$ of o_1 in $D2L(dev_3) = L3$. The previous tracking record says that the object o_1 was tracked at dev_1 before dev_3. So from ERDG we need to get the information from the edge, E($D2L(dev_1) = L1$, $D2L(dev_3) = L3$). The ENLD from L1 to L3 is $l_{1,3s}$ = 8m. Now with the help of Eq. (A.1) and (A.2), the $time_{start}$ = 15 - $\frac{8m}{3m/(18-15)}$ = 7.

Depending on the topological structure of the location, an object may have many $time_{out}$s from the same tracking device. For example $L4$ and $L5$ has loops where an object can circulate in the location which may results in multiple tracking records for same object from the devices $L2DIn(L4)$ and



(a) type 1 node **(b)** type 2 node **(c)** type 3 node

(d) type 4 node **(e)** type 5 node

Fig. A.3: Types of Nodes in CP symbolic locations

$L2DIn(L5)$. Based on the topological connectivity of a location we classified the nodes of the deployment graph into five types. Fig. A.3 shows the five node types. Different types of nodes and the way of deriving the exit time of objects from that node is explained next.

Node Types. ***Node type 1*** contains only one outgoing edge and the outgoing edge is labeled by dev_{v_x}. A location l falls in *Node type 1* if $L2DOut(l)$ = $\{dev_{v_x}\}$. Fig. A.3a shows an example of *Node type 1*. As the next location of this type of node has no tracking device deployed, it is certain that the object left the location through virtual tracking device dev_{v_x} which actually does not generate any tracking record. In Eq. (A.3) the $time_{out}$ of an object o_i at this type of location l_i is taken from the tracking record of o_i at $L2DIn(l_i)$ and EXLD $l_i, *e$ is taken from the tuple T_{L_{prev},l_i} of $edge(L_{prev}, l_i)$.

Node type 2 contains two outgoing edges. One outgoing edge is labeled by dev_{v_x} and another one is a loop. A location l falls in *Node type 2* if $L2DOut(l)$ = $\{L2DIn(l), dev_{v_x}\}$. Fig. A.3b shows an example of *Node type 2*. In our example, $L5$ is this type of node. Here an object can circulate within the location which generates multiple tracking records and at the end the object leaves the location through dev_{v_x}. The $time_{end}$ of the object is calculated using Eq. (A.3), where $time_{out}$ is taken and $speed$ is calculated from the last

tracking record of the object from the tracking device of that location. Suppose an object $o2$ contains a single record from $dev5$: (o_2, $dev5$, 36, 39). It means that o_2 did not circulate at *D2L(dev5) = L5* and left the location to any one of the chutes. It is not possible to know when the object actually left $L5$. However we can get the maximum possible value of $time_{end}$ with the help of EXLD from the *edge (*, L5)* which is $l_{5,*e}$ = 22m and CD for $dev5 = l_{d5}$ = 3m. So the $time_{end} = \lceil 39 + \frac{22m}{3m/(39-36)} \rceil = 61$.

Node type 3. In addition to the two outgoing edges like *Node type 2*, it has one or more edge(s) where destination locations have tracking devices deployed. A location l is considered to be *Node type 3* if $|L2DOut(l)| > 2$ and $\{L2DIn(l), dev_{v_x}\} \subset L2DOut(l)$. Here an object can circulate in the same location and it can leave the location through dev_{v_x} or other tracking devices. Fig. A.3c shows an example of *Node type 3*. In our running example $L4$ falls in this type of node. As the object may circulate within the location we take $time_{out}$ in the similar way of *Node type 2*. However, the EXLD in Eq. (A.3) depends on the destination of the object. If the object has any tracking record from $L2DOut(l) \setminus \{dev_{v_x}\}$ (where l is *Node type 3*) then the object did not leave the location l through dev_{v_x}. Otherwise it has left the location through dev_{v_x} without generating any tracking record. For the first case we take the corresponding EXLD otherwise we take the EXLD for the loop. For example, for $L4$, *L2Dout(L4) ={dev4, dev5, dev_v}* where $dev4$ = *L2In(L4)*. As the object $o1$ in Table A.1 has no tracking record from $dev5$, the object $o1$ should circulate at $L4$ and left the location through dev_{v_x} without generating any tracking record. So, the $time_{end}$ of object $o1$ from $L4$: $time_{end} = \lceil 54 + \frac{22m}{3m/(54-51)} \rceil = 76$.

Node type 4 and ***Node type 5*** do not contain any outgoing edge with dev_{v_x} in label. These two node types are very similar except that *Node type 4* contains a loop and *Node type 5* does not. Fig. A.3d and Fig. A.3e show examples of *Node type 4* and *Node type 5* respectively. In our running example L1, L2, L3 falls in *Node type 5*. As *Node type 4* contains a loop, the $time_{end}$ of an object o from a location l of *Node type 4* is calculated from the last $time_{out}$ from the tracking records like *Node type 2 and 3*. However the EXLD $l_{l,L_{nexte}}$ is taken from the *edge(L_{prev}, l)*. For *Node type 5* the $time_{out}$ is directly taken from the tracking record as there is no loop in it. The EXLD in *Node type 5* is taken similarly as in *Node type 4*. In our running example the $time_{end}$ of $o1$ from $L3$ is calculated as: $time_{end} = \lceil 18 + \frac{3m}{3m/(18-15)} \rceil = 21$.

Table A.2 shows the results after mapping from Table A.1.

4 Hotspot Queries

The hotspots can now be extracted from the tracking records after mapping into *MappingTable*. A hotspot query $HQ[q_s, q_e, \theta]$ finds the hotspots between time q_s and q_e where θ is the density threshold. In the inner part of a HQ,

Table A.2: MappingTable for Table A.1

MappingID	ObjectID	LocationID	$time_{start}$	$time_{end}$
map1	*o1*	*L1*	*2*	*7*
map3	*o1*	*L3*	*7*	*21*
map5	*o1*	*L4*	*21*	*76*

there is a density query (DQ), a count query (CQ) and a tracking record query (RQ). Fig. A.2 shows the approach for processing a hotspot query. When a $HQ[q_s, q_e, \theta]$ query is asked, the system issues a $DQ[q_s, q_e]$, the $DQ[q_s, q_e]$ then issues a $CQ[q_s, q_e]$ which issues an $RQ[q_s, q_e]$. The *RQ* gets the mapping table from the database and returns the mapping records where $[time_{start}, time_{end}]$ intersects with $[q_s, q_e]$. From the relevant records, the *CQ* counts the number of objects for each location. The *DQ* then finds the density of each location from the count results with the help of capacity of the corresponding location. The *HQ* then returns the locations with $density>\theta$. All of these queries can be combined into a single query and can be executed jointly. For a relational database the joint query becomes the following SQL statement. In the joint query, the *RQ* becomes the part of the *WHERE* condition, the *COUNT(DISTINCT ObjectID)* is used for *CQ*, the *DQ* is represented in the column list and is computed with the help of *CQ* and a *Capacity(Location)* function. The results are grouped based on location using *GROUP BY* and temporary stored in an inline view. Finally the *HQ* is completed with the help of a *WHERE* condition on the results from the inline view.

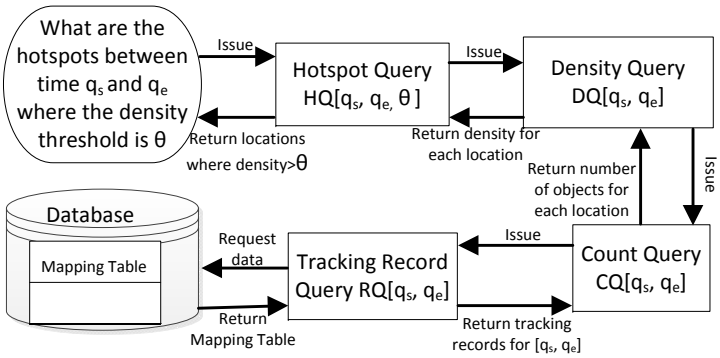


Fig. A.4: Query steps

SQL: *SELECT location, density FROM (SELECT location, (COUNT (DISTINCT ObjectID) (t_e-t_s)) Capacity (location) * 100 AS density FROM MappingTable m WHERE ($m.t_{in}$ BETWEEN t_s AND t_e) OR ($m.t_{out}$ BETWEEN t_s AND t_e) OR (t_s BETWEEN $m.t_{in}$ AND $m.t_{out}$) GROUP BY location) WHERE*

$density > \theta$

5 Conclusion and Future Work

We proposed an approach to extract the hotspots from indoor tracking data. We developed a graph-based model for mapping the tracking records with the semantic location so that it is possible to know the entry and exit times of an object at a constrained path symbolic location. Then the mapping records are used for hotspots extraction. The mapping records are also very useful for other kind of analyses e.g., stay duration, travel time estimation etc.

Future work will be to model more complex indoor topologies for mapping the same information we did for constrained path. An indexing technique for efficient query processing can be developed. Hotspot query for online indoor tracking data will be another relevant future work.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

Paper B

Finding Dense Locations in Indoor Tracking Data

The paper has been published in the
Proceedings of the 15th IEEE Mobile Data Management (MDM), Brisbane, Australia, pp. 189–194, 2014. The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1109/MDM.2014.29

Abstract

Finding the dense locations in large indoor spaces is very useful for getting overloaded locations, security, crowd management, indoor navigation, and guidance. Indoor tracking data can be very large and are not readily available for finding dense locations. This chapter presents a graph-based model for semi-constrained indoor movement, and then uses this to map raw tracking records into mapping records representing object entry and exit times in particular locations. Then, an efficient indexing structure, the Dense Location Time Index (DLT-Index) is proposed for indexing the time intervals of the mapping table, along with associated construction, query processing, and pruning techniques. The DLT-Index supports very efficient aggregate point queries, interval queries, and dense location queries. A comprehensive experimental study with real data shows that the proposed techniques can efficiently find dense locations in large amounts of indoor tracking data.

1 Introduction

Technologies like RFID and Bluetooth enable a variety of indoor tracking applications like tracking people's movement in large indoor spaces (e.g., airport, office building, shopping mall, and museums), airport baggage tracking, items movement tracking in supply chain system, etc. The huge amount of tracking data generated by these types of systems is very useful for analyses and decision making. These analyses are useful for different kinds of location-based services, finding problems in the systems, and further improvement in the systems. Unlike GPS based positioning for outdoor systems, indoor tracking provides the symbolic locations of the objects in indoor space. Examples of symbolic locations include security and shopping areas in airports, and the different sections of rooms in museum exhibitions. In airports, the bags pass different symbolic locations in each step like check-in, screening, sorting, etc. Finding the dense or overloaded baggage handling locations helps handling bags more efficiently. For passengers moving in an airport, detecting dense locations shows where and when passengers gather, and can be used for crowd management and providing location-based passenger services.

In our previous short paper [14], we proposed a graph based model and mapping technique for capturing dense locations in constrained indoor movement only. In the present chapter, we additionally model semi-constrained indoor movement and provide technique that maps the indoor tracking records into mapping records with the entry/exit times of an object at a symbolic location. The derived mapping records can be used for finding dense locations as well as for other analyses like stay duration estimation, etc. We also propose an efficient indexing structure, the Dense Location Time Index (*DLT*-Index), which stores aggregate information like the number of objects entering, exiting, and present at a location at different timestamps or time intervals. Additionally, we provide efficient techniques for index construction and processing dense location queries (as well as point and interval queries) on large data sets, and an efficient pruning technique for the *DLT*-Index. Finally, we perform a comprehensive experimental evaluation with real data showing that the proposed solutions are efficient and scalable.

The remainder of the chapter is organized as follows. Section 2 gives the problem formulation. Section 3 describes the graph-based models and the mapping of tracking records. Section 4 presents the *DLT*-Index and the associated query processing and pruning techniques. Section 5 presents the experimental evaluation. Section 6 reviews related work. Finally, Section 7 concludes the chapter.

2 Problem Formulation

The movements of objects inside indoor space varies with the structure of the paths. Based on the path structure we categorize the indoor spaces into two categories.

Constrained Path Space(CPS): In a constrained path space (CPS), objects move continuously from one symbolic location to another. The objects cannot move freely inside the locations and the locations are in some sense one dimensional. The size of locations inside CPS is measured by length not by area.

Example of a CPS can be the conveyor belt system of an airport baggage handling system. The conveyor is divided into different symbolic locations like check-in belts, screening belts, sorter belts, etc. More detail about the constrained indoor movement can be found in [14] and about the baggage tracking process in [15].

Semi-Constrained Path Space (SCPS): In a semi-constrained path space (SCPS), the objects move more freely compared to a CPS. The objects move from one symbolic location to another and they can also stay some period of time inside the locations. The locations are two-dimensional. The size of SCPS locations is measured by area not by length.

Examples can be movement of people between rooms in office space or museums, different sections in airports etc. Fig. B.1 shows an example of an SCPS. The indoor space is divided into different symbolic locations e.g., rooms, hall ways etc. For entering and exiting a room, an object has to cross the entry/exit points (e.g., doors). Some entry/exit points are unidirectional and some are bi-directional. The arrows represent the unidirectional movements.

In our setting, the tracking devices are strategically deployed at different fixed locations inside the indoor space, e.g., each section of conveyor belts, door of a room, between sections of a hallway etc. The objects contain tags (e.g., RFID tags, Bluetooth devices, etc.,) that can be tracked by the tracking devices (e.g., RFID readers, Bluetooth access points etc.,). In Fig. B.1 the circles represent the deployment of the RFID readers and their tracking ranges. When an object comes under a tracking device's activation range, it is continuously detected by the tracking device with a sampling rate and it generates raw reading records with the form: *(trackingDeviceID, ObjID, t)*. It means that a tracking device *trackingDeviceID* detects a moving object *ObjID* in its activation range at timestamp t. A *TrackingRecord(recordID, ObjID, TrackingDeviceID,* $time_{in}$, $time_{out}$*)* table is constructed from the raw tracking sequence, where *recordID* is record identifier and $time_{in}$, $time_{out}$ respectively represent the timestamps of first reading and last reading of *ObjID* by *TrackingDeviceID* in its activation range. An example table for tracking records of

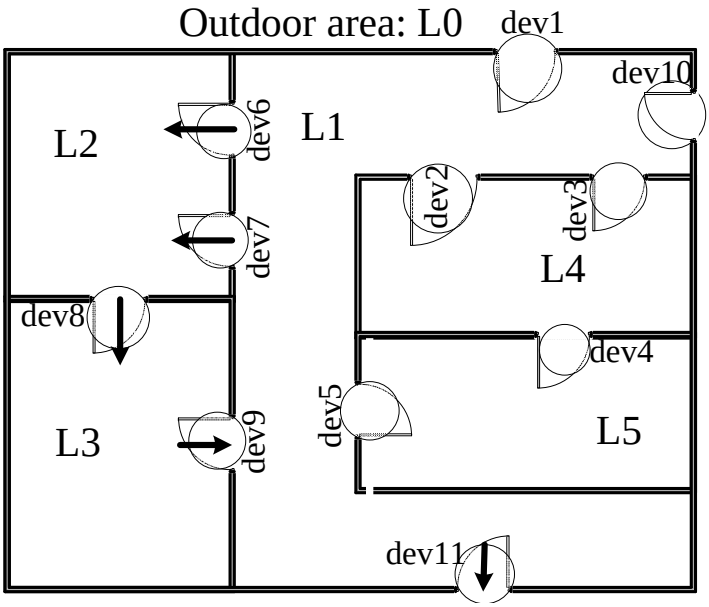


Fig. B.1: Semi-constrained space

an object o_1 at floor plan of Fig. B.1 is shown in Table B.1. Here, the record *rec3* means that o_1 is observed by device *dev3* from time 15 to 18.

Table B.1: Tracking Records of Indoor Moving Objects

RecordID	ObjID	TrackingDeviceID	$time_{in}$	$time_{out}$
rec1	o_1	*dev1*	*4*	*5*
rec3	o_1	*dev3*	*15*	*18*
rec5	o_1	*dev4*	*26*	*29*
rec8	o_1	*dev4*	*51*	*54*

Problem Definition. Let L be the set of all symbolic locations inside a large indoor space, $L = \{l_1, l_2, l_3, \ldots, l_k\}$. The *capacity* of location l_i is denoted by $c_i = capacity(l_i)$.

Definition 1 (Capacity). The *capacity* of a location l_i is the numbers of objects that can appear in l_i during a given time unit.

For example, the *capacity* of room *L*3 of Fig. B.1 can be 20 persons per 15 min.

Definition 2 (Density). Let n_i be the number of objects appearing in location l_i during the time interval, $w = [t_{start}, t_{end}]$ and $c_i = \text{capacity}(l_i)$ be the capacity of location l_i. Then the *density* of location l_i for interval w is defined as,

$d_i = \frac{n_i}{\Delta t \times capacity(l_i)} \times 100\%$, where $\Delta t = t_{end}$ - t_{start}.

Thus, the *density* shows how dense a location is, as a percentage value.

Definition 3 (Dense Location). A location l_i can be considered as a *dense location (DL)* for interval w if d_i exceeds a given threshold θ.

Definition 4 (Dense Location Query). A *dense location query (DLQ)* finds all the *dense locations* $\subseteq L$, for time interval w.

3 Semantic Location Mappings

A location is typically not fully covered by a tracking device. Moreover, a tracking record contains only the first and last times an object appeared inside the activation range of a tracking device. As a result it is not directly available when an object actually entered ($time_{start}$) and exited ($time_{end}$) the corresponding location. So mapping strategies are required for retrieving such location and timing information.

As mentioned earlier, in our previous short paper [14] we modeled constrained indoor movement like CPS and described how to convert tracking records into mapping records that contain the entry time ($time_{start}$) and exit time ($time_{end}$) of objects at different locations. For CPS, some locations contain only one tracking device deployed at any point in each of them and some adjacent locations may not contain tracking devices. An extended reader deployment graph (ERDG) was proposed that contains various topological information like entry lag distance(ENLD), exit lag distance(EXLD), covered distance(CD) etc. The timing information from the tracking records and the corresponding *CD* from the ERDG is used to calculate the speed of the object at the corresponding location. Depending on the topological structure, the nodes were categorized into 5 types. All this information helped for finding the entry and exit time of an object at a CPS location. All details can be found in [14].

In an SCPS, a tracking device is deployed at each entry and exit points of a location. Moreover, the movement can be both uni-directional and bi-directional. For example in Fig. B.1 the door containing *dev1* can be used for both entering and exiting the location *L1* and the door with *dev6* can be used only to enter *L2*. Since in SCPS the tracking devices are deployed in the entry and exit points of a location, it is easier to find the entry and exit times of objects for this type of locations from the tracking records compared to CPS. Thus, calculating the speed of the objects is not needed and thus the parameters *ENLD, EXLD* and *CD* are not useful for SCPS. However, we also need to consider that there can be multiple entry and exit points in an SCPS location. A Reader Deployment Graph (RDG) [46] is used for modeling SCPS. Fig. B.2 shows the *RDG* for the floor plan given in Fig. B.1. The *RDG* is a directed graph where each symbolic location is represented as node and

the connection between the locations are represented as edge. Each edge is labeled by the tracking devices deployed between the locations. For mapping, we define a mapping function called *Dest:* $\{l, d\} \rightarrow l'$, where $l, l' \in L$ and $d \in D$. The function*Dest(l, d)* returns node l' from the graph where d is the label for the edge $E\ (l, l') \in E$. It means that, an object traveling from l to l' should be detected by d. For mapping, initially the location of an object o is assumed to be in the outdoor location *L0*. Then each tracking record of o is accessed and it is determined where o is entering with the help of *Dest* function. Then the $time_{in}$ in the tracking record becomes $time_{start}$ and the $time_{in}$ of the next tracking record becomes $time_{end}$ for the entered location. For example in Table B.1, record *rec1* represents that o_1 was tracked by *dev1* from time 4 to 5. The initial location of o_1 is considered as $L0$. From the graph in Fig. B.2, *Dest(L0, dev1) = L1*. From tracking record, $time_{start} = time_{in} = 4$. However, the $time_{end} = time_{in}$ *from rec3* = 15. So the mapping record says that o_1 was at $L1$ from time 4 to 15. Table B.2 shows the SCPS mapping results for Table B.1.

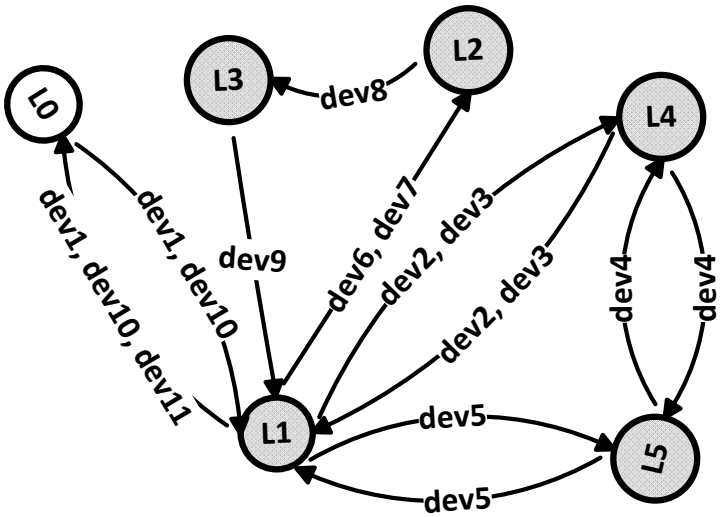


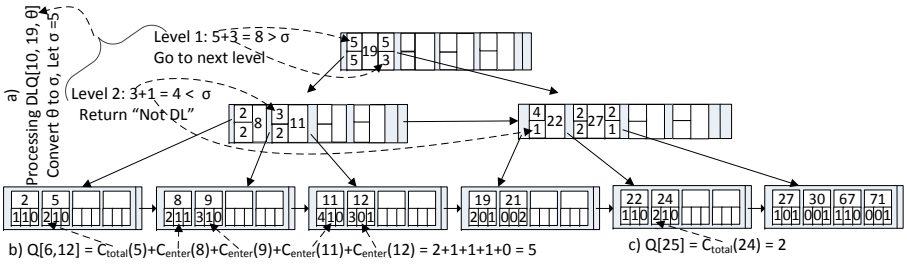
Fig. B.2: Graph based model for SCPS

Table B.2: MappingTable for Table B.1 considering SCPS

MappingID	ObjectID	LocationID	$time_{start}$	$time_{end}$
map1	*o1*	*L1*	*4*	*15*
map3	*o1*	*L4*	*15*	*26*
map5	*o1*	*L5*	*26*	*51*
map5	*o1*	*L4*	*51*	...

MapID	ObjID	LocID	$time_{start}$	$time_{end}$
r1	o_1	*L1*	*2*	*7*
r2	o_2	*L1*	*5*	*11*
r3	o_{18}	*L1*	*8*	*20*
r4	o_{15}	*L1*	*9*	*18*
r5	o_{16}	*L1*	*11*	*20*
r6	o_{12}	*L1*	*67*	*70*
r7	o_{19}	*L1*	*22*	*26*
r8	o_{20}	*L1*	*24*	*29*

(a) Example mapping records of location $L1$



(b) DLT-Index

Fig. B.3: Example mapping records and DLT-Index

4 Efficient Dense Location Extraction

The dense locations can now be extracted from the mapping records in *MappingTable*. As seen, a mapping record contains the entire stay of an object inside a location. For a CPS location like baggage on a conveyor belt, it is not common an object appears multiple times at same location whereas it is common for an SCPS location. In our setting, an object visiting a location multiple times is treated as multiple objects for density computation for that location, since a reappearance contributes to the density.

A DLQ has to access aggregate information for a given time interval from a large amount of data from the mapping table. We develop an indexing technique for that, where the temporal indexing part is motivated by the *time index* described in [31]. Instead of indexing all the records of the mapping table we index all the intervals of each location using separate trees and store aggregate values instead of pointers to the leaf nodes. Moreover, unlike the B^+-tree we link the nodes of intermediate levels. All these improvements let us avoid accessing detailed data records and further offers significant pruning opportunities. This new index structure is called the *Dense Location Time Index* (DLT-index).

4.1 The DLT-Index.

In the *DLT*-Index for each location, we maintain a separate tree, called *DLT*-tree. Let us consider a set of mapping records at location $L1$ shown in Fig. B.3a. Fig. B.3b shows the *DLT*-Index constructed for the data given in Fig. B.3a. In our *DLT*-tree, each leaf node entry at time point t_i is of the form: $\langle t_i, C_i \rangle$, where $C_i \langle c_{total}, c_{enter}, c_{exit} \rangle$ is a tuple with some aggregate information of the objects valid during $[t_i, t_i^+)$ where t_i^+ is the next indexed time point, c_{total} and c_{enter} are the total number of objects available and entered at t_i respectively and c_{exit} = total number of objects that exited at t_i-1. Besides, each non-leaf entry at time point t_i contains a tuple $C_{nl}(t_i)$ which is of the form:$\langle c'_{total}, c'_{enter} \rangle$ and their values can be described as:

1. For the left-most entry of a level: c'_{total} = c'_{enter} = total number of objects entered or available until t_i-1.

2. For the entries other than left-most entry: c'_{total} and c'_{enter} are the total number of objects available and entered during interval $[t_i^-, t_i)$ respectively, where t_i^- is the immediate left entry of t_i in the same level.

In addition to C_{nl}, the right-most entry t_i of each non-leaf level also contains another tuple $C_r \langle c''_{total}, c''_{enter} \rangle$ where c''_{total} = total number of objects available from t_i to max time stamp in the tree and c''_{enter} = total number of objects entered from t_i to the max time stamp in the tree.

Tree construction and insertion of a new entry in the *DLT*-tree is very similar to the B^+-tree. The time points are keys and aggregate information C are the data values. The value of C for the leaf levels can be precomputed or can be computed while inserting. After constructing the tree, the aggregate information of the non-leaf entries and the links between the non-leaf nodes have to be established.

4.2 Tree Construction from Historical Data

During the tree construction the historical data are indexed. Let the set of all intervals available in the data set be $I = \{I_1, I_2, ..., I_n\}$. For an interval I_i, the value of $I_i.t_s$ and $I_i.t_e$ represents the start and end time respectively. Additionally the value of $I_i^{\dashv} = I_i.t_e + 1$ represents the next timestamp after $I_i.t_e$. The *DLT*-Index has to index all the time points of P where P can be defined as follows:

$$P = \{t_i | \exists I_j \in I((t_i = I_j.t_s) \vee (t_i = I_j^{\dashv}))\}$$

For example, considering the table in Fig. B.3a, the points that need to be indexed are *P = {2, 5, 8, 9, 11, 12, 19, 21, 22, 24, 27, 30, 67, 71}*. As seen for *MapID* $r1$, the $time_{end}$ =7 is not included in P as we index the next timestamp

of 7 which is 8. Also at time point 8, o_{18} has entered the location (*MapID r*3). Before the tree construction all the time points of P are sorted in ascending order into P_s. Each of the time point $t_i \in P_s$ additionally contain a C_i where c_{enter} and c_{exit} are directly known while getting each element of P_s from the data set. For example, at the time point 8: c_{enter} = 1 and c_{exit} = 1, at time point 21: c_{enter}=0 and c_{exit} = 2 as two objects has $time_{end}$ at 21-1 = 20. The c_{total} at time point t_i is calculated by Eq. (B.1). For example, at initial stage C_0 = $\langle 0,0,0 \rangle$. For the first time point 2, c_{enter}=1 and c_{exit} = 0. So, c_{total} = 0+1-0=1. As a result C_1 = $\langle$1,1,0$\rangle$. Similarly for time point 5 c_{total} = 1+1-0=2 and C_2 = $\langle$2,1,0$\rangle$. The complete list of C_i can be found in the leaf nodes of Fig. B.3b.

$$c_{total}(t_i) := c_{total}(t_{i-1}) + c_{enter}(t_i) - c_{exit}(t_i) \tag{B.1}$$

The insertion of the time points are now same as B^+-tree where the time points are keys and C are the data values. In addition to the insertion, the nodes at the non-leaf levels have to be linked like *level 2* of Fig. B.3b. Moreover, for the entries of the non-leaf levels, the values of C_{nl} and C_r have to be calculated. Maintaining the aggregate information in the leaf nodes gives advantage for such calculation. For any non-leaf entry t_i, if it is the left-most entry in its level then the C_{nl} can be calculated as follows:
$C_{nl}(t_i).c'_{total}$ = $C_{nl}(t_i).c'_{enter}$ = $c_{total}(t_1)$ + $\sum_{j=2}^{i-1} c_{enter}(t_j)$

Besides, for any non-leaf entry t_i, if it is not the left-most entry in its level then the C_{nl} can be calculated as follows:

Let t_i^- be the entry immediately before t_i in its level and k be the position of t_i^- in the leaf node entries. Then,

$$C_{nl}(t_i).c'_{total} = c_{total}(t_k) + \sum_{j=k+1}^{i-1} c_{enter}(t_j)$$
$$C_{nl}(t_i).c'_{enter} = \sum_{j=k}^{i-1} c_{enter}(t_j)$$

Similarly for the right-most entry t_i of a non-leaf level, C_r can be calculated as follows:

$$C_r(t_i).c''_{total} = c_{total}(t_i) + \sum_{j=i+1}^{max} c_{enter}(t_j)$$
$$C_r(t_i).c''_{enter} = \sum_{j=i}^{max} c_{enter}(t_j)$$

where t_{max} represents the maximum value of the indexed time point in the leaf level.

For example in Fig. B.3b at level 2, 8 is the left-most entry. So $C_{nl}(8).c'_{total}$ = $C_{nl}(8).c'_{enter}$ = $c_{total}(2)$ + $c_{enter}(5)$ = 1+1 = 2. Considering 22 of level 2 which is not the left-most entry, $C_{nl}(22).c'_{total}$ = $c_{total}(11)$ + $c_{enter}(12)$ + $c_{enter}(19)$ + $c_{enter}(21)$ = 4+0+0+0 = 4. In the example tree 27 is the right-most entry of level 2. The value of $C_r(27).c''_{total}$ = $c_{total}(27)$ + $c_{enter}(30)$+ $c_{enter}(67)$ +$c_{enter}(71)$ = 1+0+1+0 = 2.

4.3 Query Processing

Here we will discuss how an aggregate point and interval query can be executed from our DLT-Index and then show the dense location query processing with and without pruning.

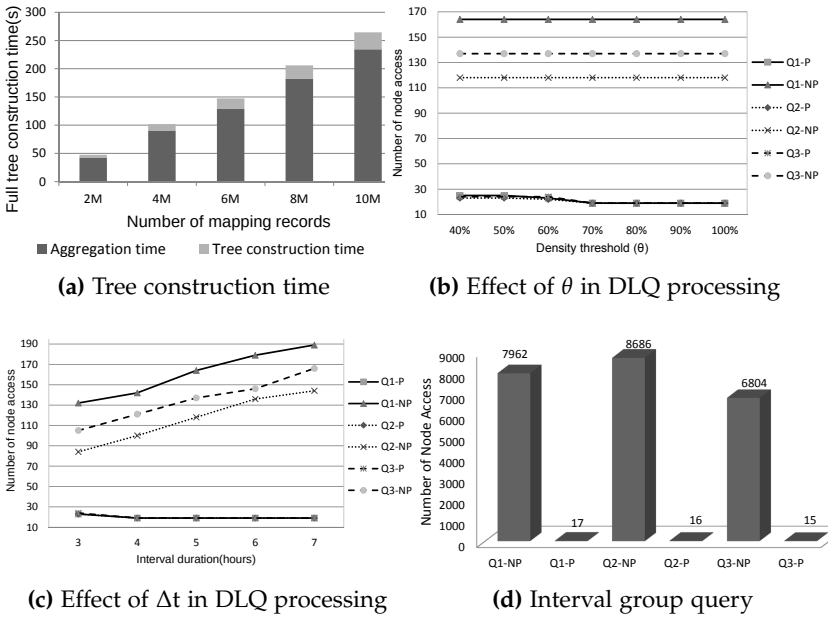
Aggregate query: There are two types of aggregate queries: a) point queries b) interval queries. A point query finds the total number of objects at a particular time *point*. An interval query finds the total number of objects in a particular time *interval*. For processing a point query $Q[q_t]$, a B^+-tree search is performed for finding the appropriate leaf node for q_t. Then it finds the last entry $q_a \leq q_t$ in the node and returns $C_{total}(q_a)$ as the result. For example consider a point query $Q[25]$. The B^+-tree search will find the node shown in Fig B.3bc. From the resulting node, entry 24 is the last entry which is ≤ 25. So the query will return $c_{total}(24) = 2$. Conversely for an interval query $Q[q_s, q_e]$, it will first process a point query $Q[q_s]$ and take $C_{total}(q_a)$. However, instead of returning the result it continues further processing and finds the last entry $q_b \leq q_e$ from the leaf nodes. For this purpose it may has to access consecutive leaf nodes one after another. If $q_a = q_b$ then it returns $C_{total}(q_a)$ as result. Otherwise it adds all the C_{enter} for each entry after q_a until q_b. For example consider an interval query $Q[6, 12]$. The B^+-tree search will find the leaf node containing $\{2, 5\}$ as shown in Fig. B.3bb. Then it finds the entry 12 which is the last entry ≤ 12. So the calculation of the result will be: $Q[6, 12] = c_{total}(5) + c_{enter}(8) + c_{enter}(9) + c_{enter}(11) + c_{enter}(12)$ = 2+1+1+1+0 = 5.

Dense Location Query: The naive approach of processing a $DLQ[q_s, q_e, \theta]$ over the DLT-Index is to just get the results of an interval query $Q[q_s, q_e]$ for each of the locations, compute the density from the result, and then determine which locations are DLs. However, for each of the cases, the query has to access the leaf nodes and then compute the aggregation. The access of the nodes of the leaf level as well as other level can be reduced by our pruning technique. As described the DLT-Index contains some aggregate information in the non-leaf node entries for pruning purpose. We first introduce the following two observations.

Observation 1 (obs 1): If the number of objects in an interval i_1 is less than a threshold k, a smaller interval i_2 that is fully covered by i_1 must have less than k objects.

Observation 2 (obs 2): If the number of objects in an interval i_1 is greater than a threshold k, a larger interval i_2 that fully covers i_1 must have more than k objects.

In our pruning technique the first observation helps to prune out the non-leaf levels, whereas the second observation helps to prune at the leaf level. Note that the cases covered by obs 1 and 2 are mutually exclusive. In a $DLQ[q_s, q_e, \theta]$ the given θ is a density threshold, not a number of objects. Be-



(a) Tree construction time **(b)** Effect of θ in DLQ processing

(c) Effect of Δt in DLQ processing **(d)** Interval group query

Fig. B.4: Experimenting with different aspects of the *DLT*-Index.

fore starting the query processing we convert θ to a number of object threshold σ_i for each location l_i. The value of σ for a location L_i for a $DLQ[q_s, q_e, \theta]$ can be derived from the density formula which is given below:

$\sigma(L_i, \theta) = \frac{\Delta t \times capacity(L_i) \times \theta}{100}$, where $\Delta t = q_e - q_s$.

While processing a $DLQ[q_s, q_e, \theta]$, the value of σ for each location has to be calculated. The query has to traverse the tree of each location. The way of accessing the next level is similar to B^+-tree search for q_s. At each non-leaf level l_i of the *DLT*-tree of a location Loc_i, the smallest interval $[t_a, t_b)$ that can cover $[q_s, q_e]$ has to be found. After that, the total number of objects $n(l_i)$ during $[t_a, t_b)$ is calculated from the C_{nl} stored in all the entries between t_a and t_b in level l_i. The value of $n(l_i)$ is compared with σ. If $n(l_i) < \sigma$ then Loc_i is marked as 'Not DL' based on *obs 1* and further processing of the tree is pruned. However, if $n(l_i) \not< \sigma$ then the next level is accessed and continue further processing in the similar way. If the query reaches the leaf level then it starts calculating the total number of objects during $[q_s, q_e]$. During the calculation, at each step it compares the sum with σ and if $sum > \sigma$ then based on 'emphobs 2 the location Loc_i is marked as *DL*. As a result it skips the access of the next entries and nodes and avoids further calculation. If the condition $sum \leq \sigma$ then the full sum is calculated to decide whether Loc_i is DL or not.

Consider Fig. B.3ba, where the processing of *DLQ[10, 19, θ]* is shown for location L_1. Let $\sigma = 5$ be derived from θ. At the root level, the smallest interval that covers $[10, 19]$ is [*min, max*) where *min* and *max* are the starting and ending time points of the tree respectively. For *min* the $C_{nl}.c'_{total}$ from the left-most entry is taken, and for max $C_r.c''_{enter}$ is taken from the right-most entry of the current level. So in our case the total number of objects during this period is $C_{nl}(19).c'_{total}$ + $C_r(19).c'_{enter}$ = 5+3=8. As $8 \not< \sigma$, the query has to go to the next level based on B^+-tree search for 10. In this level $[8, 22)$ is the smallest interval that covers $[10, 19]$. Now the total number of objects for the period is $C_{nl}(11).c'_{total}$ + $C_{nl}(22).c'_{enter}$ =3+1=4. As $4 < \sigma$ is true, the location $L1$ is not a DL during $[10, 19]$. As a result, it does not need to access the next level and do further processing for $L1$. Now consider another query $DLQ[6, 12, \theta]$ and let $\sigma = 3$. During the processing of this query on the tree for $L1$ shown in Fig. B.3b, the pruning with *obs 1* does not work and the query will access the leaf level. However, the query does not have to fully compute the total number of objects during $[6, 12]$ as the sum up to time point 9 is: 2+1+1=4 (Fig. B.3bb) and $4 > \sigma$. So, it can be deduced that the location is a DL during $[6, 12]$ without further processing.

5 Experimental Study

Experimental Setup: We implemented the mapping for CPS and pre-computed the aggregate information from the mapping records using C# and the DLT-Index using C. For all SQL queries, we use a leading RDBMS. The experiments were conducted on a laptop with an Intel Core i7 2.7 GHz processor with 8 GB main memory. The operating system is Windows 7 64 bit.

We use real RFID based baggage tracking data from the transfer system of terminal-3 of Copenhagen Airport (CPH). It has 11 CPS locations with 11 RFID readers deployed. After filtering some erroneous records there are 2.1 M tracking records for 220 K distinct bags collected during Dec 21, 2011 to Dec 02, 2013. As we do not have the exact values, the parameters ENLDs and EXLDs are generated. As most of the locations contain loops, after converting 2.1 M tracking records there are 787K Mapping records produced.

DLT**-Index**: The page size was set to 4KB and each entry of the DLT-tree was 20 bytes. This yields 204 entries per node.We use the above mapping table produced from the airport baggage tracking data and scaled it to 10M mapping records. For scaling, Max(ObjId) is added with existing ObjId for uniqueness and random time between 50 to 120 seconds are added with both $time_{start}$ and $time_{end}$ for each record. The semi-real data contains 3M distinct objects and total 17.3 M distinct time points distributed to the 11 symbolic locations.

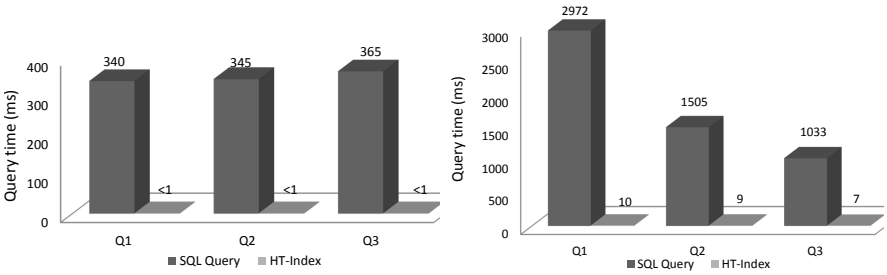
Tree Construction: Before the tree construction the aggregate information

C for each time point is precomputed. Then the trees are constructed for the time points including the aggregate information. Fig. B.4a shows the effect of number of mapping records on the full tree construction time along with the aggregate information generation time. It shows that the tree construction time increases linearly with the increase in number of mapping records. Generating the aggregate information takes good amount of time as it has to access each record for finding distinct time points, sort and union them and compute the value of c_{total}, c_{enter} and c_{exit} at each time point.

Query Processing: Three random *DLQs* are generated for semi-real data and are processed with both pruning (P) and without pruning (NP). The node access for each of them is reported in Fig. B.4b and B.4c. In all cases, pruning accesses much fewer nodes compared to without pruning. In Fig. B.4b the query time interval is kept fixed and the density threshold(θ) is changed. It shows that, for without pruning, a query has to access same number of nodes regardless of change in θ as it always has to compute total number of objects for full interval. However, for pruning, the number of node access decreases with increase in θ as the number of dense location decreases and σ increases (effect of pruning with *obs 1*) and at some point it becomes constant. In Fig. B.4c, θ is kept fixed and the time interval length is changed. It seems that for without pruning, node access increases with increase in interval length as the query has to cover more time points to complete the interval. However, for pruning, node access decreases and become constant at some point. Here, any of the observation (*obs 1* or *obs 2*) can be the reason depending on the number of dense locations. The query is effected by *obs 1* if number of dense location decreases and by *obs 2* if number of dense location increases. We also process three random queries that finds number of objects at each location for one month large interval with a given condition on the aggregate results (like group by with having condition in SQL) to see the effect of pruning on larger interval in the *DLT*-Index. Fig. B.4d shows that the pruning technique access significantly less number of nodes compared to without pruning.

We randomly generate 3 point queries and also 3 interval queries that find number of objects at each location for a given time interval. The interval queries are for one year large interval. The queries are processed by both the *DLT*-Index and the relational DBMS. At first we have processed the queries inside the DBMS without creating any indices in the *MappingTable*. Then we have created both clustered index and non-clustered index on different columns and on combination of columns and processed the queries. However, the queries were fastest without any indices. So, we have reported the query processing durations without an index. For the fair comparison, the first query processing time for each query in the SQL is ignored for warm up and to load the data into main memory. Then we execute each query five times and take the average query time. Fig. B.5a and B.5b shows that the point query processing in *DLT*-Index is more than 340 times faster and some

cases the interval query processing time is around 300 times faster.



(a) Point query **(b)** Interval group. query

Fig. B.5: *DLT*-Index vs. RDBMS

6 Related Work

Several papers address density queries and hot route queries on road networks [45, 55] in outdoor spaces. There are spatio-temporal data access methods for point and interval queries on temporal dimension in outdoor settings [70, 73] and in indoor settings [47]. These indexing techniques are capable to efficiently access individual records for a range of locations as well as for a time period. However, for a DLQ, aggregate information for each location has to be accessed. Our *DLT*-Index itself contains aggregate information and avoids accessing individual data records. There are also works for indexing spatio-temporal data for aggregate queries [64]. The aRB-tree [64] stores the aggregate information in the tree nodes. However, it counts same objects multiple times if the objects remain in the same location in several timestamps during the query interval. This problem is solved by an approximate approach in [72]. Unlike to the traditional range and point queries where the query generally does not access all the locations, the DLQ has to access all the locations' information to determine whether each location is a DL or not. So like the aRB-tree, we also maintain a separate tree for each location except the *R*-Tree part. The distance functions in the symbolic space are very far from the Euclidean distance, so the MBRs cannot approximate the distances well. Thus, the MBRs used in aRB-tree is not applicable in our case. As mentioned earlier our temporal indexing structure is motivated by [31], but we extend it to a symbolic spatio-temporal space and we additionally maintain necessary aggregate information in nodes to facilitate counting distinct objects for a time interval. The aggregate information stored in the non-leaf nodes helps achieve effective pruning in processing DLQs as well as interval and group by with having conditions.

7 Conclusion

We proposed an approach to extract the dense locations from indoor tracking data. We developed a graph-based models for mapping the tracking records with the semantic location so that it is possible to know the entry and exit times of an object at a symbolic location. Then we proposed an indexing technique, the Dense Location Time Index (*DLT*-Index), that indexes aggregate information with the time points. We also proposed efficient techniques for index construction and query processing and pruning, for dense location queries on the *DLT*-Index. Our experimental evaluation on large amounts of real data shows that the *DLT*-Index can process queries efficiently and much faster than an RDBMS. The *DLT*-Index is also useful for general time interval indexing for efficient processing of queries like the number of distinct records for a specified time point or time interval.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

ISSN (online): 2246-1248
ISBN (online): 978-87-7112-529-0

AALBORG UNIVERSITY PRESS